Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Д. И. Самаль, В. В. Колонов

СТРУКТУРНАЯ И ФУНКЦИОНАЛЬНАЯ ОРГАНИЗАЦИЯ ЭВМ. ОСНОВНЫЕ БЛОКИ

Лабораторный практикум

для студентов специальности

1-40 02 01

«Вычислительные машины, системы и сети»

всех форм обучения

Библиотека БГУИР

Минск БГУИР 2011

УДК 004.3 (076.5)
ББК 32.973.2я73
С17

Рецензент:
заведующий лабораторией логического проектирования
Объединённого института проблем информатики
Национальной академии наук Беларуси,
доктор технических наук П. Н. Бибило

Самаль, Д. И.

С17 Структурная и функциональная организация ЭВМ. Основные блоки : лаб. практикум для студ. спец. 1-40 02 01 «Вычислительные машины, системы и сети» всех форм обуч. / Д. И. Самаль, В. В. Колонов. – Минск : БГУИР, 2011. – 44 с.: ил.

ISBN 978-985-488-667-1.

В лабораторном практикуме рассмотрены типовые задания на лабораторные работы, выполняемые в рамках дисциплины «Структурная и функциональная организация ЭВМ». Для каждой работы приводится соответствующий теме теоретический материал.

Особое внимание уделено вопросам использования для моделирования схем системы автоматизированного проектирования Quartus 7.1.

УДК 004.3(076.5)
ББК 32.973.2я73

ISBN 978-985-488-667-1

© Самаль Д. И., Колонов В. В., 2011
© УО «Белорусский государственный университет информатики и радиоэлектроники», 2011

Библиотека БГУИР

Содержание

Библиотека БГУИР

Введение

Настоящий лабораторный практикум содержит варианты заданий к 6 лабораторным работам, посвящённым вопросам проектирования составных частей электронных вычислительных машин. Каждая из работ включает теоретический материал, что позволяет без использования дополнительной литературы спроектировать требуемый блок вычислительной машины.

Так как разработка систем подобной сложности в отведённое для лабораторных работ время без использования средств САПР практически невозможна, то лабораторный практикум основывается на использовании системы автоматизированного проектирования фирмы Altera – Quartus v.7.1.

Для облегчения выполнения лабораторных работ первые три из них сопровождаются подробным пошаговым руководством пользователя названной выше САПР.

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №1
Освоение навыков работы в САПР Quartus 7.1

1.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1.1. Создание файла блок-диаграммы с описанием разрабатываемого модуля

Для создания простейшего блока, изображенного на рис. 1.1, необходимо выполнить следующие действия:

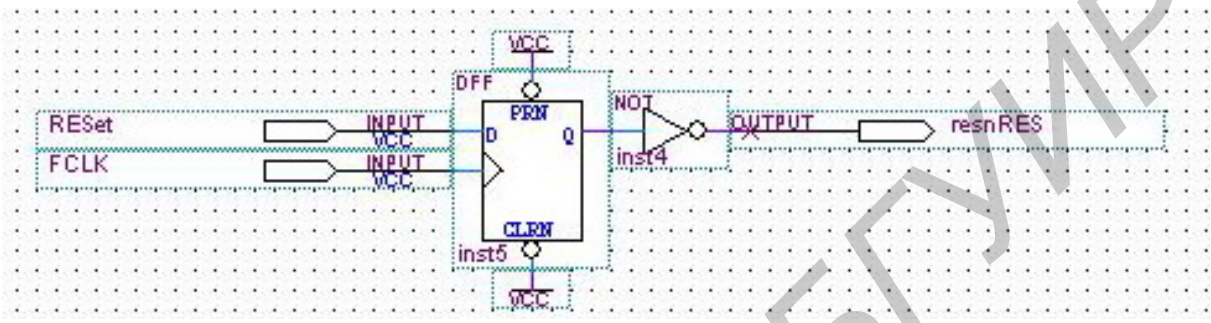


Рис. 1.1. Разрабатываемый блок resclk

а) создать новый файл командой *File→New* из основного меню либо щелчком по соответствующей иконке в меню (рис. 1.2), или быстрой комбинацией клавиш Ctrl+N;

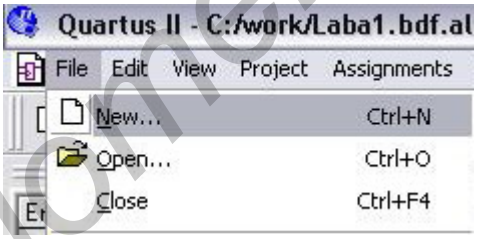


Рис. 1.2. Создание нового файла

б) выбрать тип создаваемого файла. В данном случае это будет – **Block-Diagram/SchematicFile**.

Новому файлу автоматически присваивается имя **Block<#>.bdf**;

в) созданному файлу необходимо присвоить имя и добавить его в проект.

Мастер создания проектов запускается автоматически при сохранении файла с требуемым именем (если проект не был до этого создан). Мастер (рис. 1.3) предлагает указать директорию для сохранения файлов проекта (желательно отдельную), имя проекта (которое может быть любым, в данном случае используется Lab1), а также вершину иерархии проекта (название схемы, в данном случае используется resclk). После заполнения этих полей в проект можно добавить уже существующие файлы и сразу нажать *Finish*, оставив остальные опции по умолчанию.

Библиотека БГУИР

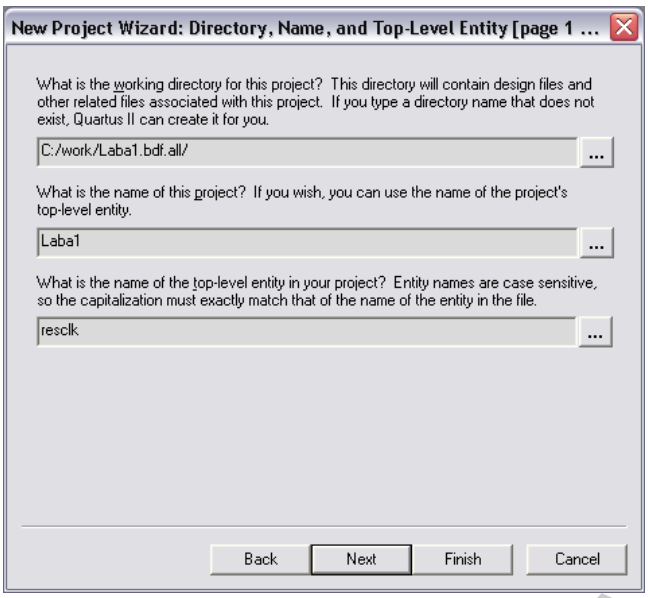


Рис. 1.3. Окно мастера создания проектов

ВАЖНОЕ ЗАМЕЧАНИЕ: *Все действия (компиляция, моделирование и т. д.) производятся со схемой, находящейся в вершине иерархии проекта (Top-Level Entity). Изменить вершину иерархии можно в Настройках проекта (пункт Assignments→Settings в меню (рис. 1.4) или комбинация Ctrl+Shift+E) в разделе General.*

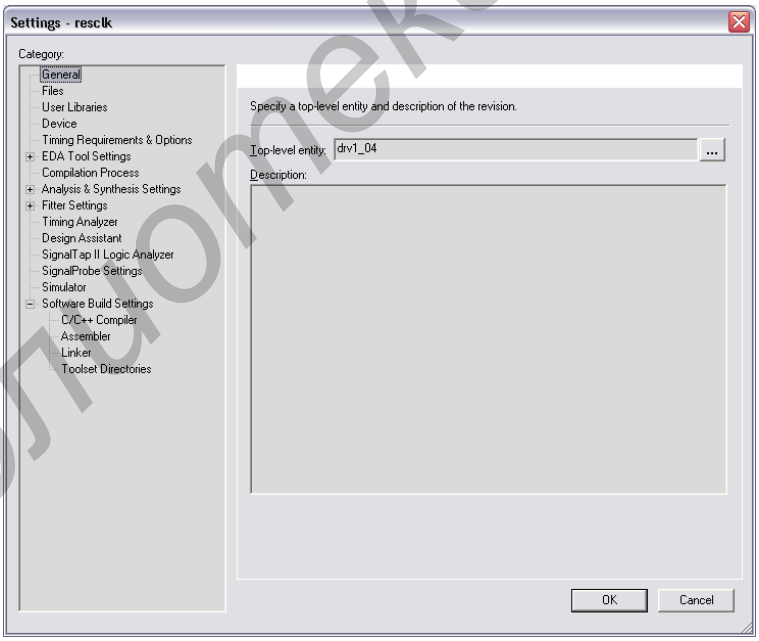


Рис. 1.4. Указание вершины иерархии в настройках проекта

Также вершину можно изменить в окне браузера проекта в разделе *Hierarchy* или *Design Units*, выбрав опцию *Set As Top Level Entity* в выпадающем по щелчку правой кнопкой мыши меню;

г) создать непосредственно саму схему. В данном случае схема **resclk** содержит элементы двух типов: схемные примитивы (входные и выходные

Библиотека БГУИР

константы – пины (**pin**))**,** триггер **DFF,** источники сигналов **VCC** и проводные соединения между примитивами – линии или шины связи (**node** и **bus** соответственно).

Символы схемных примитивов хранятся в библиотеках систем элементов САПР АЛЬТЕРА. Для вызова этих библиотек необходимо в произвольном месте bdf-листа произвести двойной щелчок левой кнопкой мыши и выбрать из соответствующей папки библиотеки нужный вам элемент;

д) произвести соединение выходов элементов в соответствии со схемой. Соединение элементов производится либо простым наложением друг на друга, либо с помощью линий связи. Переименование названий входов и выходов осуществляется двойным щелчком мыши по названиям. Имена проводникам даются после их выделения (когда они выделены и отображаются синим цветом, можно писать имя).

Провода с одинаковыми именами можно не соединять. Например, в схеме startgc (рис. 1.5) провод resn в нижней части схемы коммутировать необязательно.

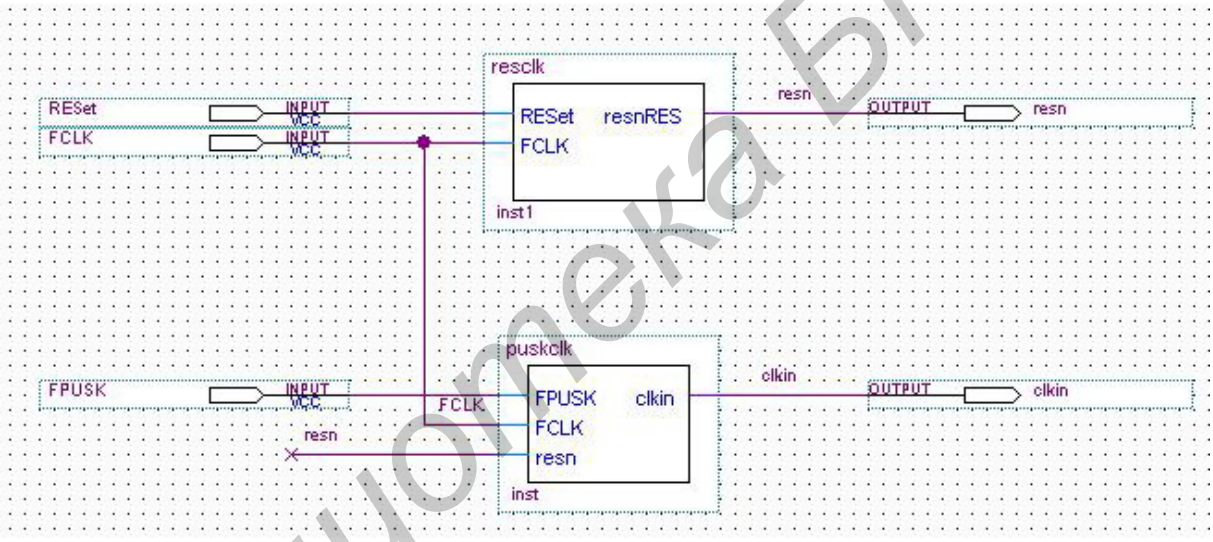


Рис. 1.5. Соединение проводов с одинаковыми именами

1.1.2. Компиляция собранного модуля

После сборки схемы проводится ее компиляция либо с помощью команды меню *Tools→Compiler Tool* и дальнейшего нажатия *Start*, либо нажатием на соответствующую иконку в меню (рис. 1.6). Не забудьте поместить схему в вершину иерархии проекта перед ее компиляцией.

Рис. 1.6. Кнопка компиляции на панели инструментов

Необходимо дождаться полной компиляции (все показатели на 100 %), после чего будет выведено количество ошибок и предупреждений. Более под-

Библиотека БГУИР

робно о найденных ошибках и предупреждениях можно узнать в окне сообщений. Также можно посмотреть рекомендации по устранению ошибок и предупреждений, выбрав Help в меню по щелчку правой кнопкой мыши на соответствующей ошибке/предупреждении в окне сообщений.

1.1.3. Моделирование схемы

Моделирование схемы производится только после успешной компиляции (без ошибок).

1. Необходимо создать новый файл временных диаграмм для моделируемой схемы *(Vector Waveform File, *.vwf)*

Выбираем в меню создание нового файла *Ctrl+N*. В открывшемся диалоговом окне на закладке *Other Files* выбираем *Vector Waveform File* (рис. 1.7).

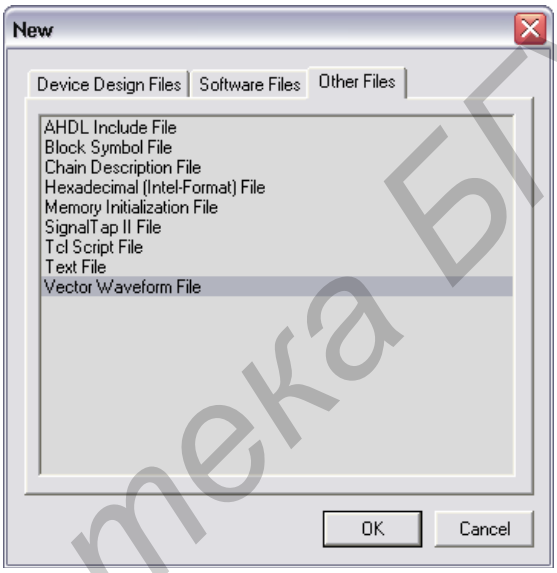


Рис. 1.7. Создание файла временных диаграмм

После нажатия *OK* откроется непосредственно сам файл (рис. 1.8).

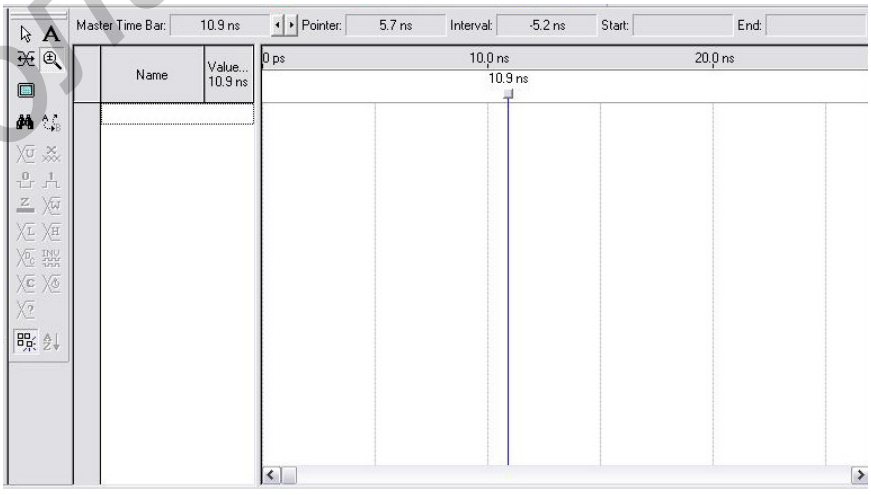


Рис. 1.8. Вид файла временных диаграмм в САПР Quartus

Библиотека БГУИР

2. Далее необходимо добавить входы и выходы из моделируемой схемы. В левой области располагаются входы, выходы и другие элементы, а в правой их изменения во времени.

Двойной щелчок в левой области вызывает диалог добавления элементов для отображения (рис. 1.9).

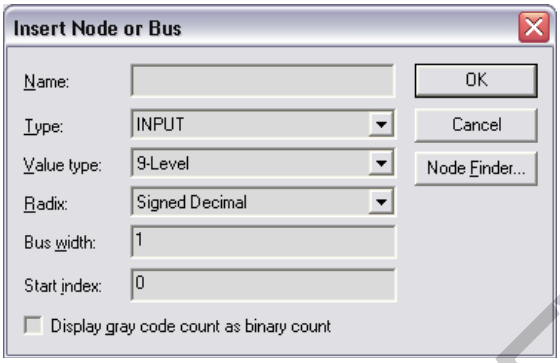


Рис. 1.9. Диалог добавления элементов для отображения

Можно добавлять элементы, вручную указывая их названия в пункте *Name*, но удобнее это делать напрямую из схемы. Для этого нажимаем кнопку *Node Finder*. В открывшемся окне (рис. 1.10) в разделе *Filter* лучше всего выбрать *Pins:all*, удостовериться, что в разделе *Looking* указана моделируемая схема, и затем нажать кнопку *List*. Стоит обратить внимание на то, что элементы могут быть не видны до перекомпиляции схемы. После того как найденные элементы отобразились в разделе *Nodes Found*, нужные из них необходимо переместить в раздел *Selected Nodes* с помощью соответствующих кнопок или двойным щелчком мыши. После нажатия *OK* выбранные элементы появятся в левой области файла.

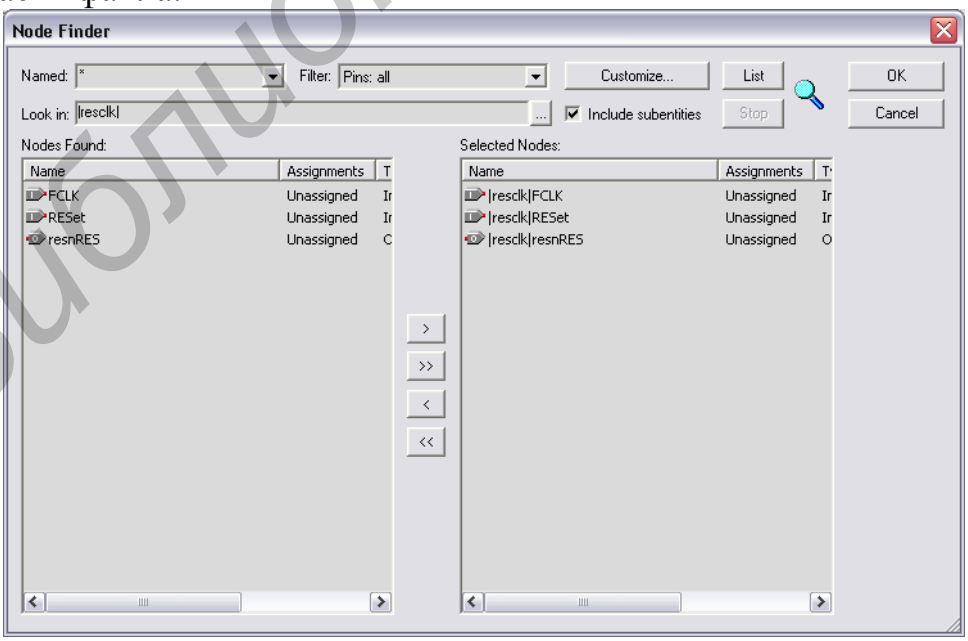


Рис. 1.10. Диалог поиска элементов

Библиотека БГУИР

3. На следующем этапе для выбранных элементов задаются значения, изменяемые во времени.

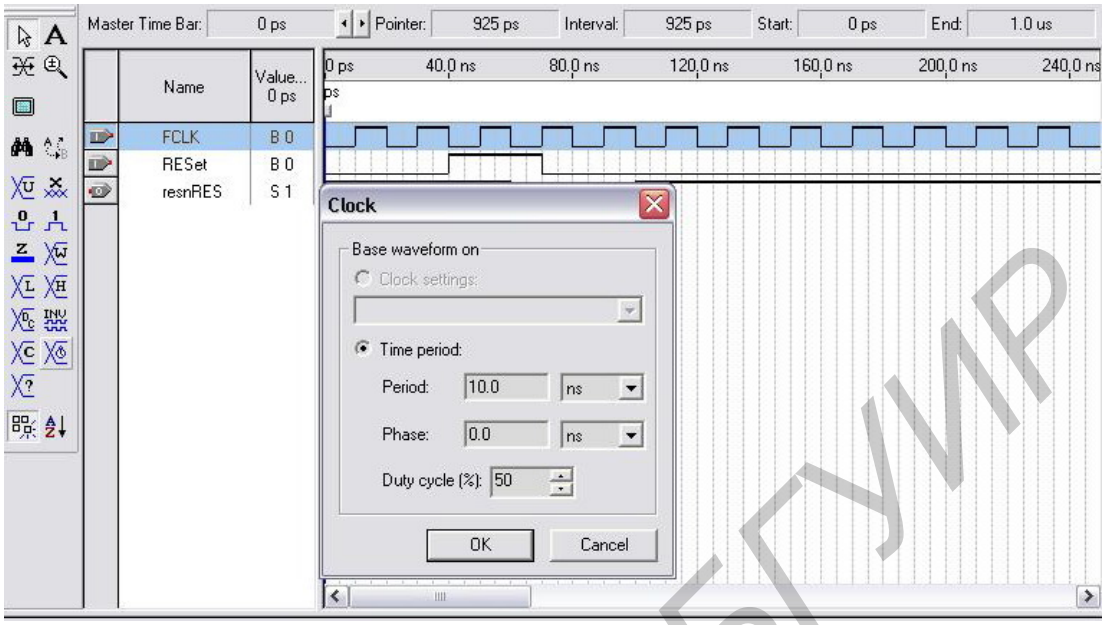


Рис. 1.11. Задание формы сигнала с помощью инструментов

Задание значения на определенном временном интервале осуществляется выделением этого интервала инструментом *Selection Tool* путем проведения по интервалу указателем с зажатой левой кнопкой мыши. На панели инструментов (рис. 1.11) находятся различные инструменты, которые могут быть полезны при задании сигналов различной формы. В частности, для задания тактирующего сигнала *Clock* удобно использовать инструмент *Overwrite Clock*.

Кроме того, сигналы можно копировать комбинациями *Ctrl+C* и *Ctrl+V* как из текущего файла, так и из любого другого файла временных диаграмм.

4. Полученную диаграмму необходимо сохранить (желательно, чтобы ее название совпадало с названием схемы).

5. После этого можно приступать непосредственно к моделированию.

Для первого моделирования удобно использовать *Simulator Tool* меню Tools, поскольку в нем можно убедиться, что используется именно тот файл временных диаграмм, который нужен, и настроить некоторые опции. При повторном моделировании можно воспользоваться командой *Processing→StartSimulation Ctrl+I* или нажать на соответствующую иконку в меню.

В разделе *Simulation input* указывается файл временных диаграмм. Галочка *Overwrite simulation input file with simulation results* означает, что полученные для выходных пинов значения будут записаны в файл временных диаграмм **.vwf*, это удобно, например, для дальнейшего копирования этих значений в другие файлы **.vwf*.

Библиотека БГУИР

6. Результат моделирования можно посмотреть либо в файле отчета (кнопка *Report* в окне симулятора, которое показано на рис. 1.12), либо в самом файле временных диаграмм, если указанная галочка была выставлена.

Рис. 1.12. Окно симулятора

На рис. 1.13 приведен результат моделирования схемы resclk. По этому результату можно проверить, правильно ли вы собрали и промоделировали схему.

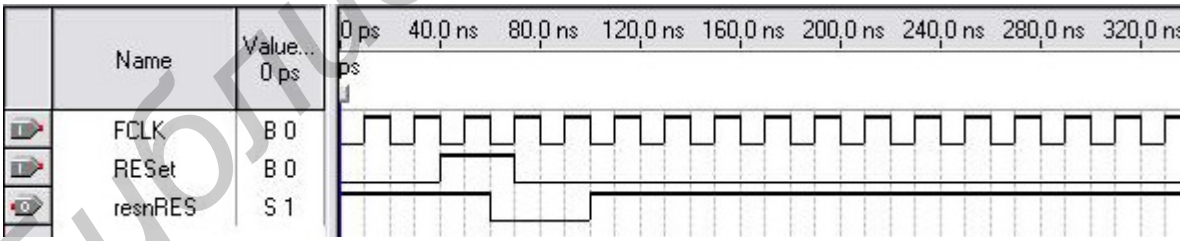


Рис. 1.13. Результаты моделирования

1.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1.2.1. Создать bdf-файл и описать указанный по варианту элемент.

1.2.2. Скомпилировать полученный файл.

1.2.3. Промоделировать собранный блок. При этом задаваемые входные воздействия должны по максимуму перебирать все варианты

Для показа выполнения необходимо продемонстрировать bdf-файл и результаты моделирования.

Библиотека БГУИР

Таблица 1.1

Варианты заданий к лабораторной работе №1

№ варианта	Реализуемый элемент
1	Регистр
2	Схема сдвига
3	Сумматор
4	Вычитатель
5	Мультиплексор
6	Демультиплексор
7	Шифратор
8	Дешифратор

ЛАБОРАТОРНАЯ РАБОТА №2
Управляемый генератор синхросигналов

2.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1.1. Использование синхросигналов в проектах САПР Quartus

Все сигналы, использующиеся для синхронизации чтения/записи в регистры, память и для других устройств, будем называть синхросигналами, или тактирующими сигналами (clock).

В результате компиляции проекта предыдущей лабораторной работы при использовании синхронных элементов вы, наверняка, получили следующее предупреждение:

Warning:"Found pins function in gas undefined clocks and/or memory enables".

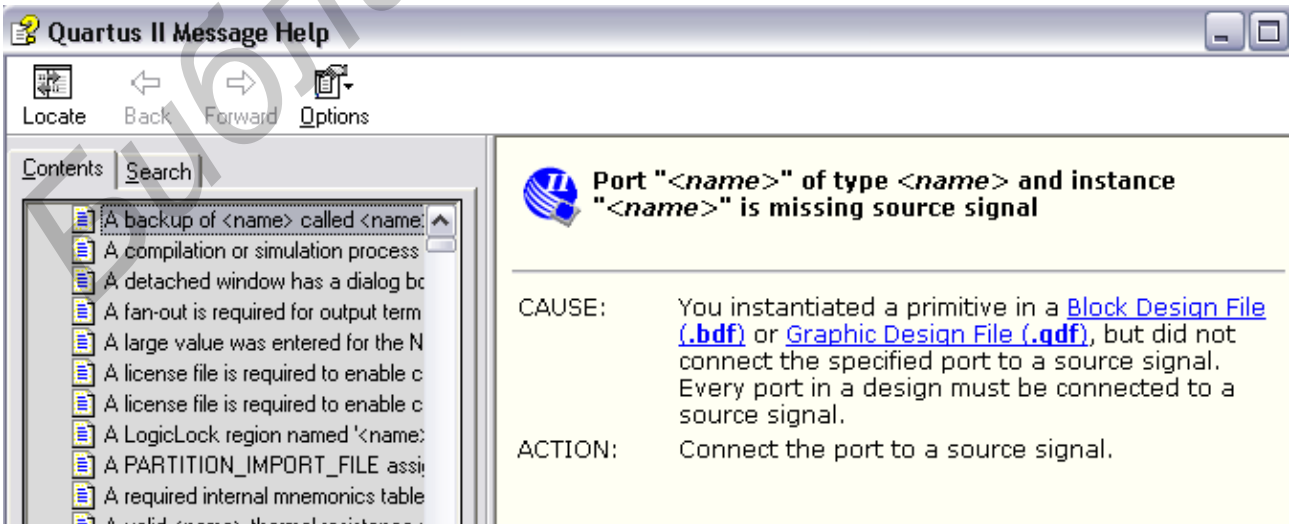


Рис. 2.1. Получение помощи по предупреждениям и ошибкам

Библиотека БГУИР

То есть система определила, что некоторые пины используются для синхронизации, но в свойствах проекта они не определены как синхросигналы. Об этом система дает пояснение в окне помощи (рис. 2.1), которое можно вызвать по данному и другим предупреждениям. Окно помощи вызывается по правому щелчку мыши в окне Processing из пункта меню Help.

CAUSE: The Timing Analyzer found one or more pins that are functioning as undefined clocks in the design. The Quartus II software identifies a pin that feeds the clock input of a register that does not have a Clock Settings timing assignment as an undefined clock.

ACTION: Altera recommends that you create clock settings and assign the clock settings to the pin(s) that are functioning as clocks in the design. If you do not want the Quartus II software to treat a pin feeding the clock input of a register as an undefined clock, you can turn on the Not a Clock assignment for the pin.

Если не наложить временные ограничения, то Quartus не будет знать, с какой максимальной частотой эти сигналы будут переключаться.

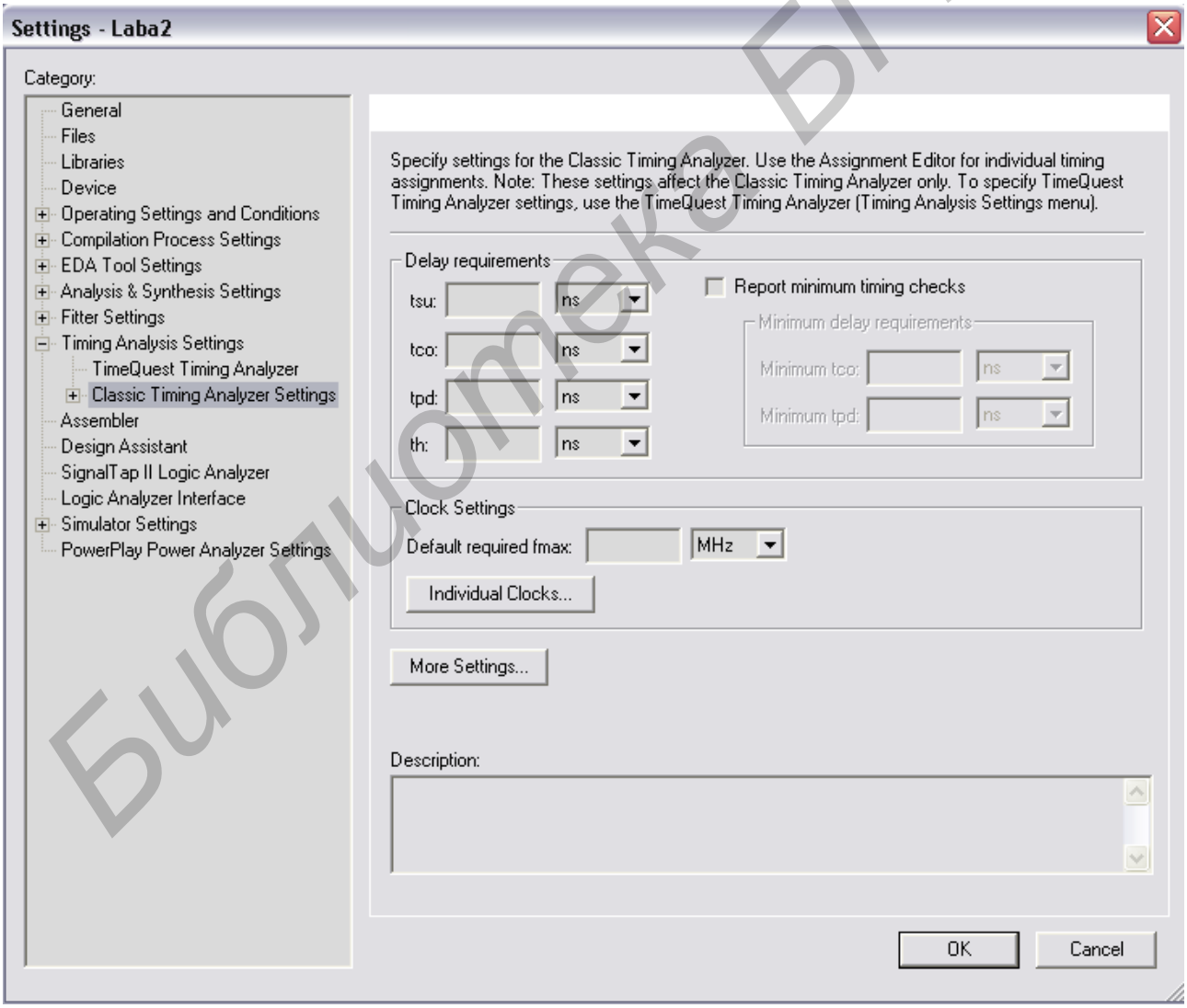


Рис. 2.2. Настройки временного анализатора в настройках проекта

Библиотека БГУИР

В случае отсутствия временных ограничений *timing constraints* на синхросигнал Quartus оптимизирует пути распространения сигналов для всех синхросигналов в системе. Если существует всего один сигнал, явно определенный как синхросигнал с ограничениями, оптимизация будет начинаться с него; если несколько – оптимизация будет производиться для всех сигналов в порядке уменьшения степени жесткости ограничений.

Поэтому в том случае, когда в проекте всего один синхросигнал, временные ограничения необязательны, но рекомендуются, поскольку в противном случае система будет выдавать предупреждение на каждый сигнал, зависящий от этого синхросигнала.

Настроить временные ограничения можно в меню *Assignments→Timer Analysis Settings* или непосредственно в настройках проекта открыть соответствующую ветку. Мы будем использовать *ClassicTimerAnalyzer*, поэтому надо убедиться в том, что выбран именно этот анализатор. После этого можно перейти к ветке *ClassicTimerAnalyzerSettings* (рис. 2.2).

В настройках можно указать требования к задержкам, а также создать настройки для синхросигналов. Нас будет интересовать последнее, для этого нажатием на *Individual Clocks* перейдем к настройкам синхросигналов (рис. 2.3).

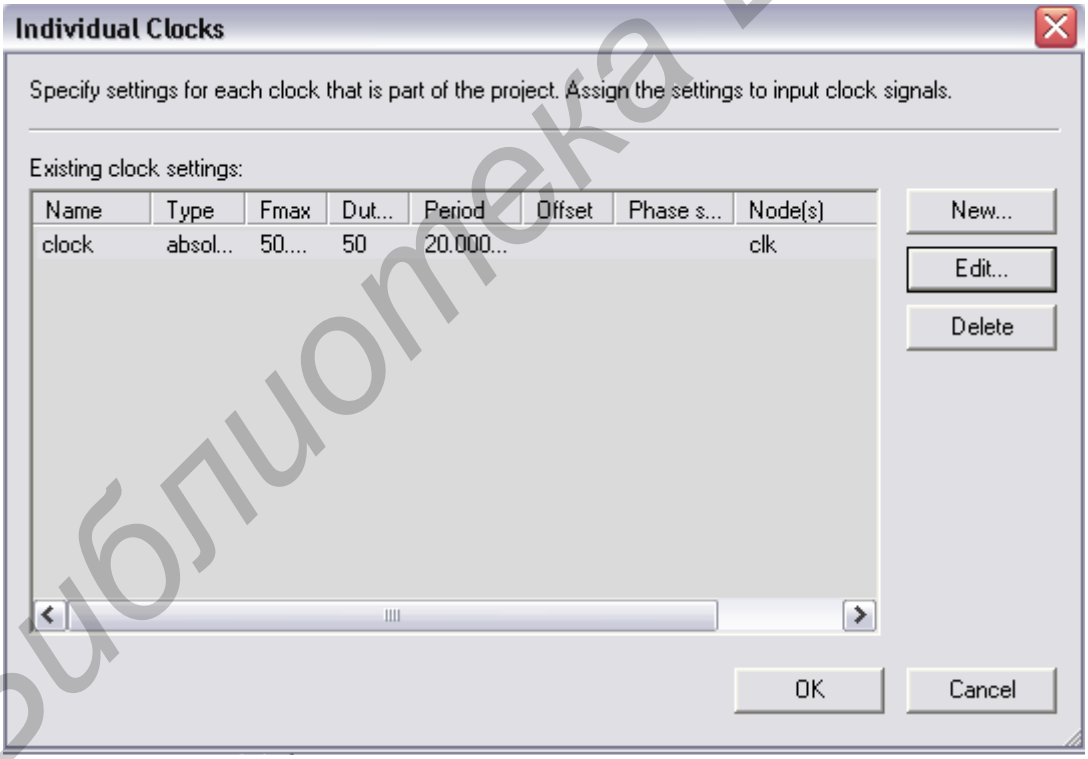


Рис. 2.3. Настройки синхросигналов

Здесь можно добавлять/изменять/удалять шаблоны настроек. Рассмотрим добавление нового шаблона (рис. 2.4).

В шаблоне необходимо указать имя шаблона в поле *Clock settings name* и сигнал, к которому этот шаблон будет применяться в поле *Applies to node*. Конкретные параметры синхросигнала можно указать двумя способами: с нуля или

Библиотека БГУИР

основываясь на уже существующих шаблонах. В первом случае необходимо выбрать пункт *Independent of other clock settings* и задать максимальную частоту *fmax* и коэффициент заполнения *Duty Cycle* – отношение длительности импульса к промежутку между импульсами. Во втором случае необходимо указать шаблон, на основе которого будет формироваться новый, в поле *Based on* и указать требования к заимствованным настройкам (рис. 2.5). Там можно указать множитель частоты, сдвиг, инверсию и др.

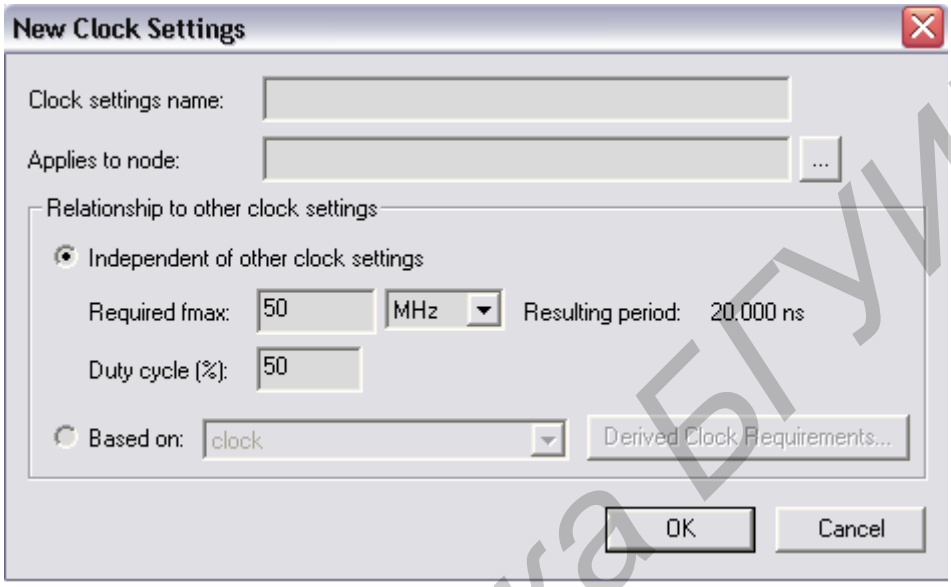
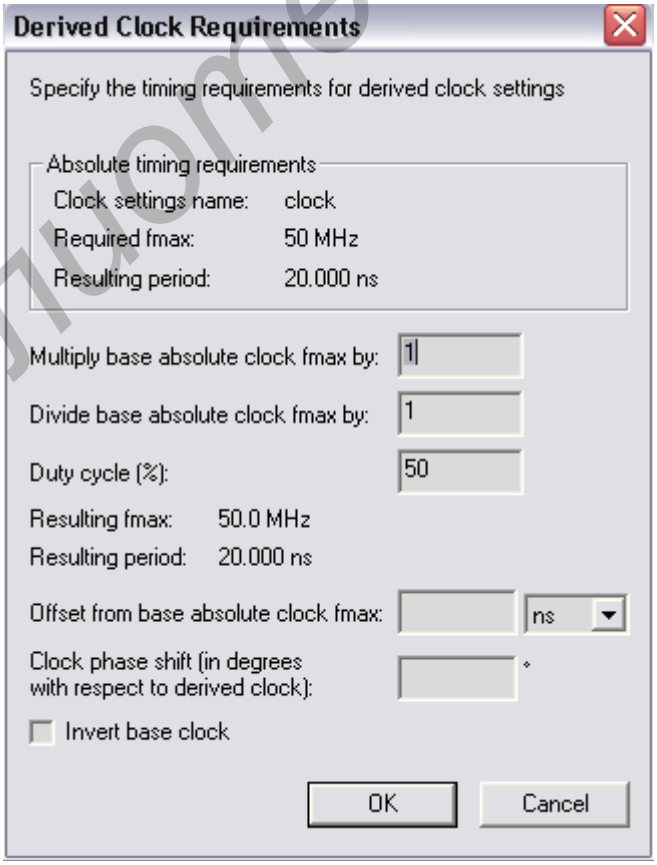


Рис. 2.4. Добавление нового шаблона настроек



Библиотека БГУИР

Рис. 2.5. Требования к заимствованным настройкам

Существующие шаблоны можно использовать при моделировании. При вводе синхросигналов в файле временных диаграмм при использовании инструмента *Clock* можно указать шаблон и тогда к сигналу будут применены настройки из шаблона (рис. 2.6). Для сигнала будут заданы соответствующая частота, коэффициент наполнения и другие настройки, если они были указаны.

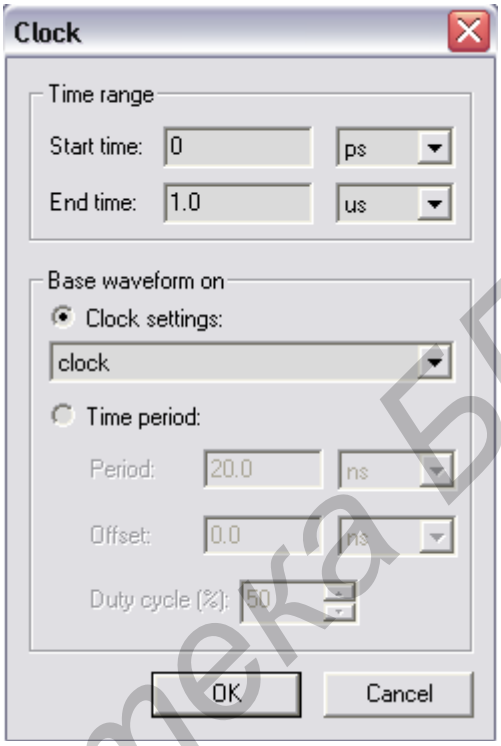


Рис. 2.6. Использование настроек синхронизации в инструменте *Clock*

2.1.2. Проектирование генератора синхросигналов

Сгенерированные сигналы предполагается использовать для синхронизации работы различных частей системы как единого целого. В соответствии с заданием генератор будет представлять собой блок, состоящий из двух частей: блока выработки единичного синхросигнала с заданным периодом и длиной импульса и блока выработки группы опорных сигналов. Для составных блоков предварительно создаются символы.

Создание символа выполняется командой меню *File→Create/Update→Create symbol files for current file* при открытом файле дизайна (*.bdf).

Опорные сигналы второго блока должны быть объединены в шину DCa[15..0]. Стоит сказать несколько слов об использовании шин в Quartus. Шины в проекте обозначаются утолщенными линиями. Названия шинам даются в формате **name[n..0]**, где n + 1 – разрядность шины, name – название шины. Чтобы входной или выходной пин стал шиной, достаточно использовать в его названии такую нотацию (рис. 2.7). С шины можно снять определенный бит, для

Библиотека БГУИР

этого отводится обычная тонкая линия, которая подписывается в формате **nameN** или **name[N]**, где **N** – номер разряда шины, начиная с нуля. Перед отведением от шины определенного бита ее необходимо обязательно подписать, даже если она отводится или заводится на входной/выходной пин. При моделировании с использованием шин рекомендуется использовать 16-ричную *Hexidecimal* систему счисления *Radix*, поэтому лучше сразу указать 16-ричную систему для всех пинов добавляемых в файл векторных диаграмм. Значение участку шины присваивается двойным левым щелчком или с помощью инструмента *Arbitrary Value* (рис. 2.8).

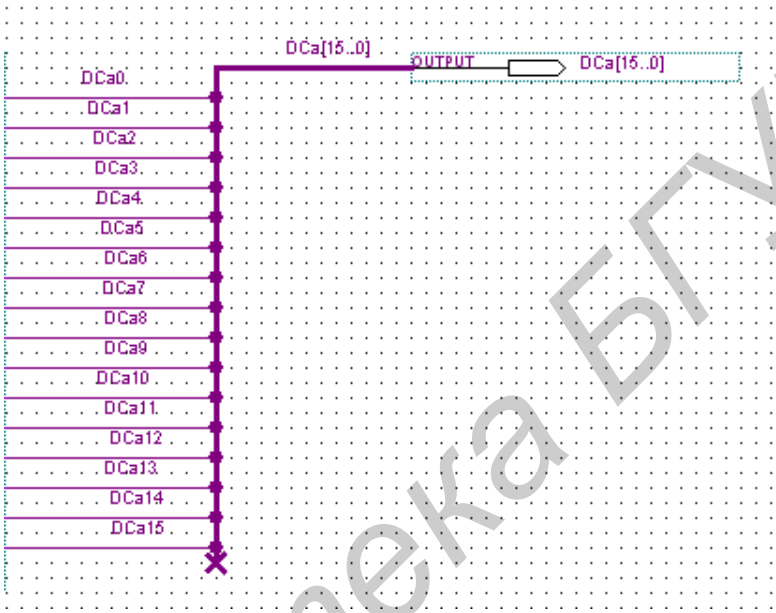


Рис. 2.7. Объединение сигналов на шину

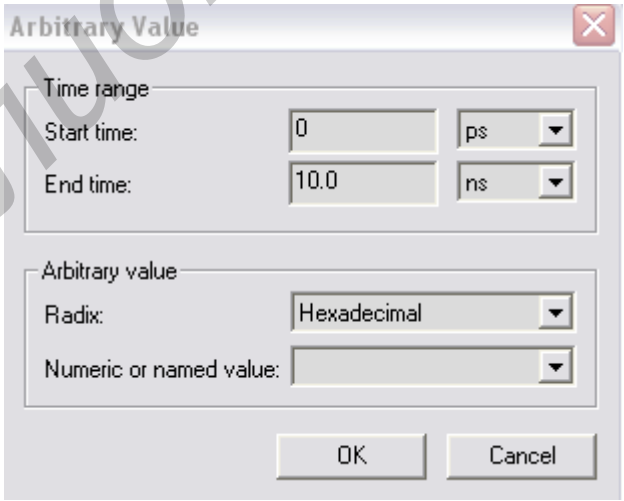


Рис. 2.8. Окно инструмента *Arbitrary Value*

Для выполнения задания по лабораторной работе могут понадобиться счетчики, дешифраторы, вычитатели, триггеры и другие элементы в зависимости от реализации генератора. В качестве этих элементов можно использовать

Библиотека БГУИР

компоненты библиотеки Quartus. Использовать готовые генераторы из библиотеки Quartus запрещается. Большой набор компонентов находится в разделе *Others→Maxplus2* библиотеки компонентов Quartus. Также вместе с этим лабораторным практикумом поставляется помощь по этим элементам – папка *libraryhelp*. Для поиска по элементам серии 74 необходимо искать в индексе пункт *74–series macrofunctions*.

2.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

2.2.1. Спроектировать и промоделировать блок, на вход которого поступает синхросигнал **clk** с частотой, заданной по варианту, а на выходе формируется сигнал с формой и указанными по варианту параметрами **m** и **n, m** и **n** – количество тактов внешнего синхросигнала **clk**. Указать временному анализатору, что **clk –** синхросигнал, и задать его в файле временных диаграмм с помощью шаблонов настроек временного анализатора.

Вариант повышенной сложности: добавить к входам блока два 8-разрядных значения, представляющие собой параметры **m** и **n**. Гарантировать получение дополнительных баллов за усложненный вариант может только оригинальное решение этого задания.

Две возможные формы сигнала изображены на рис. 2.9. В табл. 2.1 указан вариант сигнала в соответствии с рис. 2.9, а или рис. 2.9, б соответственно.

а б

Рис. 2.9. Форма выходного сигнала

2.2.2. Спроектировать и промоделировать блок, на вход которого поступает синхросигнал **clk** с частотой, заданной по варианту, а на выходе формируется 16-разрядный сигнал, представляющий собой 16 опорных синхроимпульсов **DCa (**рис. 2.10**)**. Сигнал должен представлять собой 16-разрядную шину. Форма синхроимпульсов представлена на рис. 2.10. Вариант делителя частоты **clk** для получения сигнала с частотой **clkin** указан в таблице вариантов.

2.2.3. Объединить два спроектированных блока в один, используя символы составных блоков. Промоделировать этот блок.

Семейство ПЛИС для реализации – Flex10K (изменить семейство ПЛИС *(Family)* можно в настройках проекта в ветке *Device* или в пункте меню *Assignments→Device*).

Для защиты работы необходимо продемонстрировать схему по каждому пункту и объяснить результаты моделирования п. 2.2.3.

Библиотека БГУИР

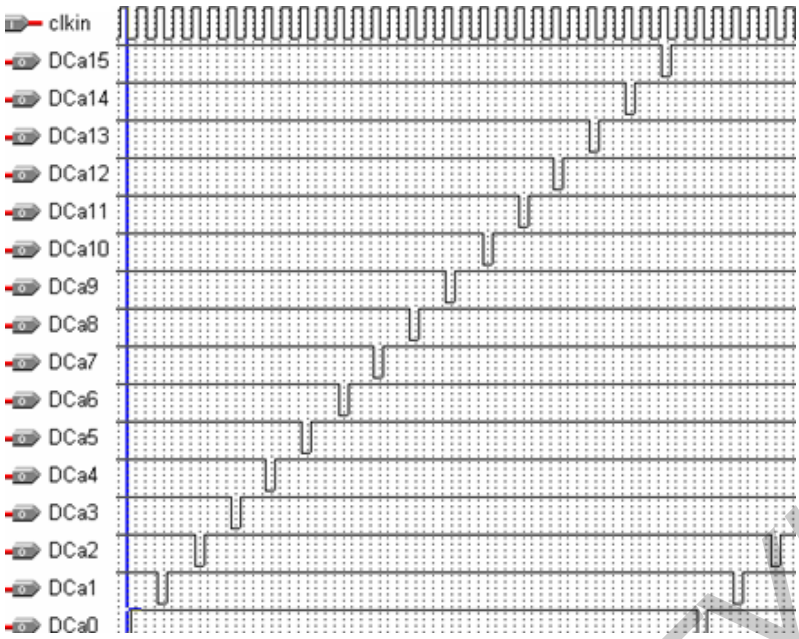


Рис. 2.10. Вид выходного опорного синхросигнала

Таблица 2.1

Варианты заданий к лабораторной работе №2

№ варианта	Частота clk, МГц	Форма сигнала (по рис.2.9)	Параметры m и n, тактов	Делитель частоты
1	50	а	12 и 3	12
2	40	б	3 и 5	10
3	25	а	11 и 4	8
4	20	б	4 и 6	6
5	12,5	а	10 и 5	4
6	10	б	5 и 7	2
7	6,25	а	9 и 6	4
8	5	б	6 и 8	6

2.3. СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

1. Задание.
2. Проектирование блока выработки единичного синхросигнала.
   2.1. Схема генератора.
   2.2. Результаты моделирования.
3. Проектирование блока опорных сигналов.
   3.1. Схема блока.
   3.2. Результаты моделирования.
4. Обобщённая схема генератора.
   4.1. Схема управляемого генератора.
   4.2. Результаты моделирования.

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №3
Использование модулей памяти

3.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ
3.1.1. Общие сведения о памяти ЭВМ

Память предназначена для фиксации, хранения и выдачи информации в процессе работы ЭВМ. Процессы чтения и записи информации определяются как процессы обращения к запоминающему устройству (ЗУ). ЗУ характеризуются:

- местом расположения (на кристалле ЦП, материнской плате, внешней памяти);
- ёмкостью;
- единицей пересылки;
- методом доступа;
- быстродействием;
- физическим типом (полупроводники, магнитный носитель, оптика);
- физическими особенностями (энергозависимая /энергонезависимая);
- стоимостью.

Емкость ЗУ характеризуют числом битов либо байтов, которое может храниться в запоминающем устройстве.

Единица пересылки обычно равна ширине шины данных (ШД) (слову), но необязательно.

Методы доступа к ЗУ:

Последовательный доступ. ЗУ с последовательным доступом ориентированы на хранение информации в виде последовательности блоков данных, называемых записями. Для доступа к нужному элементу (слову или байту) необходимо прочитать все предшествующие ему данные. Пример – магнитные ленты.

Прямой доступ. Каждая запись имеет уникальный адрес, отражающий ее физическое размещение на носителе информации. Обращение осуществляется как адресный доступ к началу записи с последующим последовательным доступом к определенной единице информации внутри записи. Пример – жесткий диск.

Произвольный доступ. Каждая ячейка памяти имеет уникальный физический адрес. Обращение к любой ячейке занимает одно и то же время и может проводиться в произвольной очередности. Пример – ОЗУ.

Ассоциативный доступ. Этот вид доступа позволяет выполнять поиск ячеек, содержащих такую информацию, в которой значение отдельных битов совпадает с состоянием одноименных битов в заданном образце. Сравнение осуществляется параллельно для всех ячеек памяти, независимо от ее ёмкости. Пример – КЭШ-память.

Библиотека БГУИР

Быстродействие ЗУ:

Время доступа. Для памяти с произвольным доступом оно соответствует интервалу времени от момента поступления адреса до момента, когда данные заносятся в память или становятся доступными. В ЗУ с подвижным носителем информации это время, затрачиваемое на установку головки записи/считывания (или носителя) в нужную позицию.

Длительность цикла памяти или период обращения (ТЦ). Понятие применяется к памяти с произвольным доступом, для которой оно означает минимальное время между двумя последовательными обращениями к памяти. Период обращения включает в себя время доступа плюс некоторое дополнительное время. Дополнительное время может требоваться для затухания сигналов на линиях, а в некоторых типах ЗУ, где считывание информации приводит к ее разрушению, – для восстановления считанной информации.

Скорость передачи. Это скорость, с которой данные могут передаваться в память или из нее. Для памяти с произвольным доступом она равна 1/ТЦ. Для других видов памяти скорость передачи определяется соотношением

$$TN = TA + N/R,$$

где TN – среднее время считывания или записи N битов; TA – среднее время доступа; R – скорость пересылки, бит/с.

Стоимость – отношение общей стоимости ЗУ к его ёмкости в битах, стоимость хранения одного бита информации.

Основная память (ОП) – единственный вид памяти, к которой ЦП может обращаться непосредственно. Основная память – ЗУ с произвольным доступом.

Основная память может включать в себя два типа устройств:

– оперативные запоминающие устройства (ОЗУ) – RAM *(Random Access Memory)*;

– постоянные запоминающие устройства (ПЗУ) – ROM *(Read Only M-ry)*.

ПЗУ обеспечивает считывание информации, но не допускает ее изменения в отличие от ОЗУ.

Традиционно понятие RAM противопоставляется ROM. Исходя из полных английских названий, можно сделать вывод, что память типа ROM не является памятью с произвольным доступом. Однако это неверно, потому как доступ к устройствам ROM может осуществляться в произвольном, а не строго последовательном порядке. И на самом деле, наименование RAM изначально противопоставлялось ранним типам памяти, в которых операции чтения/записи могли осуществляться только в последовательном порядке. В связи с этим более правильно назначение и принцип работы оперативной памяти отражала бы аббревиатура RWM *Read/Write Memory*, которая, тем не менее, практически не встречается. Стоит отметить, что русскоязычным названиям и сокращениям (ОЗУ и ПЗУ) подобная путаница не присуща.

Энергозависимые ОЗУ можно также подразделить еще на две основные подгруппы: динамическую память *DRAM – Dynamic Random Access Memory* и статическую память *SRAM – Static Random Access Memory*.

Библиотека БГУИР

3.1.2. Использование параметризированных модулей памяти в проектах САПР Quartus

В библиотеке параметризированных модулей можно найти следующие модули ROM/RAM-памяти, поддерживаемые семейством Flex10K (табл. 3.1):

Таблица 3.1

Расшифровка аббревиатур микросхем памяти

lpm_ram_dq	Parameterized RAM with separate input and output ports megafunction
lpm_ram_dp	Parameterized dual port RAM megafunction
alt3pram	Parameterized triple-port RAM megafunction
lpm_rom	Parameterized ROM megafunction
lpm_ram_io	Parameterized RAM with a single I/O port megafunction
Altdpram	Parameterized dual-port RAM megafunction
Csdpram	Parameterized cycle-shared dual port RAM megafunction

Рассмотрим некоторые из этих блоков более подробно.

Встраивание параметризированных модулей в проект, как вы уже могли заметить, можно производить с помощью мастера, который запускается автоматически после выбора соответствующего модуля из библиотеки, если выставлена галочка *Launch Mega Wisard Plugin*. Ручная настройка параметров имеет некоторые плюсы, в частности, для блоков памяти – это вход, позволяющий перевести выходы в третье состояние. При запуске мастера создается VHDL-описание фактически нового блока с указанными вами параметрами, при ручной же настройке модуль берется стандартный, а уже настройки указываются вручную в его свойствах. Задание рассчитано на использование ручных настроек. Однако это не означает, что нельзя использовать мастер, просто понадобятся дополнительные буферы для перевода линий в третье состояние, чтобы исключить конфликт шин. Советы и пояснения по работе с мастером приводятся в отдельном файле *megawisard.pdf.* Дальше приводится описание настроек, выставляемых вручную.

3.1.3. Работа с модулем lpm_rom

Находим модуль в библиотеке и добавляем его на рабочую область (рис. 3.1).

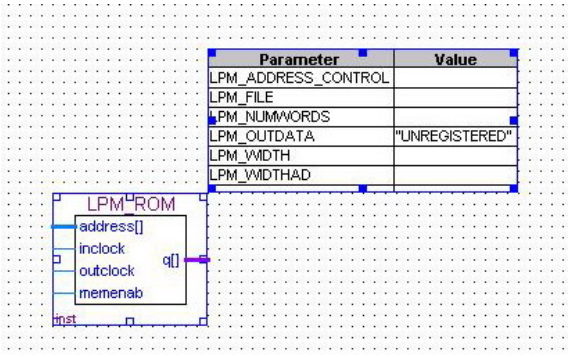


Рис. 3.1. Модуль lpm_rom

Библиотека БГУИР

Далее необходимо выбрать используемые и неиспользуемые порты. Сделать это можно в свойствах модуля на закладке «Порты» (рис. 3.2).

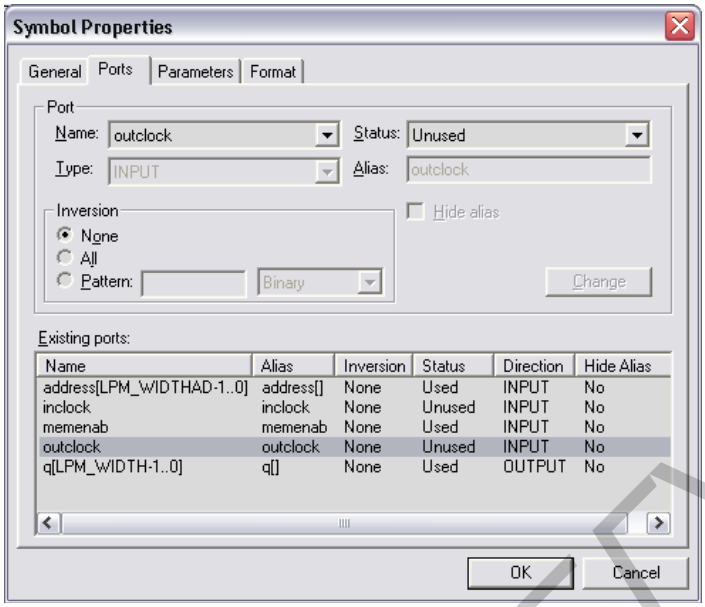


Рис. 3.2. Свойства модуля lpm_rom – закладка «Порты»

В настройках присутствуют порты *address* и *q* – соответственно порт ввода адреса и порт вывода данных. Присутствуют также два порта тактирования – *inclock* и *outclock*. Стоит обратить внимание, что если порты *inclock* и *outclock* не используются *Unused*, то ввод и вывод данных производится асинхронно. В противном случае ввод синхронизируется передним фронтом *inclock*, а вывод – передним фронтом *outclock*. Порт *memenab* позволяет перевести выходную шину в состояние высокого импеданса (отключено).

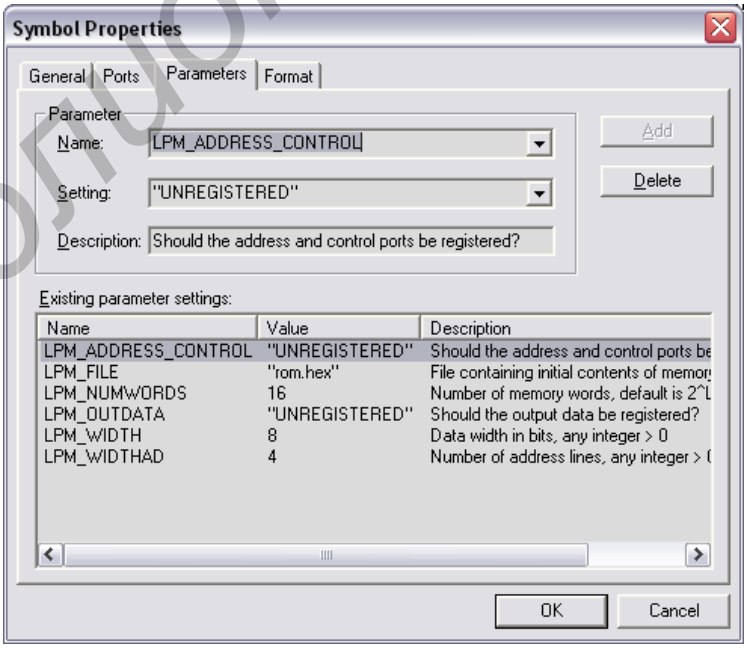
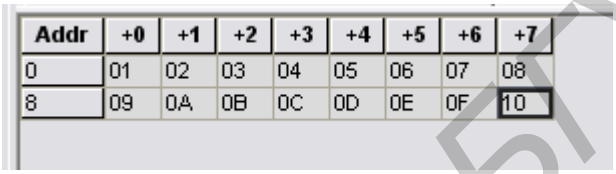


Рис. 3.3. Свойства модуля *lpm_rom* – закладка «Параметры»

Библиотека БГУИР

Переходим к закладке параметров модуля (рис. 3.3). Здесь указываются такие параметры, как разрядность адресного входа, разрядность выхода данных, количество слов в памяти, а также параметры, указывающие, присутствуют ли на входе и выходе регистры (необходимы для синхронных операций). Также здесь указывается путь и название файла инициализации памяти (так называемый файл «прошивки»). Обратите внимание на то, что все строковые параметры записываются в кавычках. По умолчанию файл инициализации памяти ищется в директории проекта. Файл может иметь формат *Memory Initialization File* (.mif) или *Hexadecimal* (Intel-Format) File (.hex). Если вы указали несуществующий файл инициализации, то перед компиляцией его необходимо создать либо в текстовом редакторе, либо в самом Quartus'е через меню New. При редактировании в Quartus'е (рис. 3.4) в меню *View→AddressRadix* и *MemoryRadix* задается система счисления для адреса и данных соответственно.



Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	01	02	03	04	05	06	07	08
8	09	0A	0B	0C	0D	0E	0F	10

Рис. 3.4. Редактирование файла прошивки в среде Quartus

После окончания настройки модуля можно попробовать промоделировать его работу. Дня этого необходимо проинициализировать память значениями из файла и получить на выходе значения по заданному адресу. Для файла *rom.hex*, содержимое которого приводилось на рис. 3.4, моделирование будет выглядеть следующим образом (рис. 3.5).

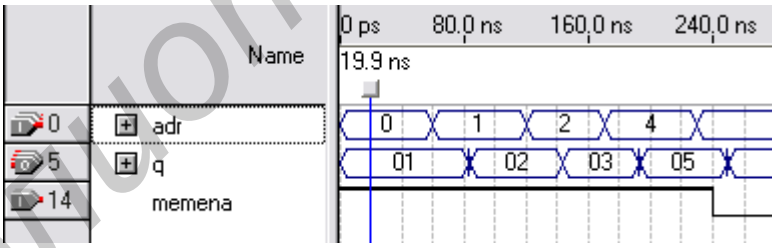


Рис. 3.5. Моделирование блока lpm_rom

Стоит также отметить тот факт, что изменение файла инициализации должно сопровождаться перекомпиляцией проекта.

Также нельзя забывать, что модуль имеет достаточно большое время срабатывания (> 20 нс). Поэтому очень часто работа с памятью является самым узким местом системы.

Таблицу истинности для данного модуля можно посмотреть в справке Quartus.

Библиотека БГУИР

3.1.4. Работа с модулем lpm_ram_io

Работа с модулем *lpm_ram_io* производится аналогично п. 3.1.3. Отличиями являются наличие входа разрешения записи и возможность выполнения операции записи в этот модуль.

Таблица истинности модуля также находится в помощи Quartus.

Особое внимание стоит уделить тактированию входа, поскольку под входом понимается не только занесение данных в модуль, но и адреса.

Результаты операций записи необходимо искать не в файле инициализации, а в отчете моделирования в пункте *Simulator→Logical Memories*.

3.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.2.1. Разработать блок, включающий в себя модули *lpm_rom* и *lpm_ram_io,* выходящие на общую шину. Продемонстрировать операции чтения из памяти. Синхронность/асинхронность операций указана по варианту.

3.2.2. Используя 8-разрядный регистр, прочитать данные из ячейки – источника памяти ПЗУ и записать их в ячейку – приемник памяти ОЗУ.

Семейство ПЛИС для реализации – Flex10K (изменить семейство ПЛИС *Family* можно в настройках проекта в ветке Device или в пункте меню *Assignments→Device*).

Таблица 3.2

Варианты заданий к лабораторной работе №3

№ варианта	lpm_rom		lpm_ram_io		Ячейка – источник	Ячейка – приемник
	Ввод	Вывод	Ввод	Вывод		
1	Синхр.	Асинхр.	Синхр.	Синхр.	1	8
2	Асинхр.	Асинхр.	Синхр.	Асинхр.	2	7
3	Синхр.	Синхр.	Синхр.	Асинхр.	3	6
4	Асинхр.	Синхр.	Синхр.	Синхр.	4	5
5	Синхр.	Асинхр.	Синхр.	Асинхр.	5	4
6	Асинхр.	Асинхр.	Синхр.	Синхр.	6	3
7	Синхр.	Синхр.	Синхр.	Асинхр.	7	2
8	Асинхр.	Синхр.	Синхр.	Синхр.	8	1

3.3. СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

1. Задание.
2. Проектирование блока памяти.
   2.1. Схема блока.
   2.2. Содержание памяти до операций ввода/вывода.
   2.3. Содержание памяти после операций ввода/вывода.
   2.4. Результаты моделирования.

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №4
Считывание, декодирование и выполнение команд. Способ адресации операндов в командах

4.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ
4.1.1. Классификация команд в ВМ

Алгоритм, написанный пользователем программы, в конечном счете реализуется в виде машинных команд. Под **командой** понимают совокупность сведений, представленных в виде двоичных кодов, необходимых процессору для выполнения очередного шага. В коде команды для сведений о типе операции, адресной информации о нахождении обрабатываемых данных, а также для информации о месте хранения результатов выделяются определенные разряды (поля).

Множество реализуемых машиной действий образует её **систему команд**. Система команд часто определяет области и эффективность применения ВМ. Состав и число команд должны быть ориентированы на стандартный набор операций, используемых пользователем для решения своих задач.

Существует несколько различных классификаций команд в вычислительных машинах (ВМ). В частности, можно выделить следующие типы команд:

– команды пересылки данных;
– команды арифметической и логической обработки;
– команды работы со строками;
– команды SIMD;
– команды преобразования;
– команды ввода /вывода;
– команды управления системой;
– команды управления потоком команд.

Команды пересылки данных

Наиболее распространённый тип машинных команд. Примеры: передача данных между ЦП и ОП, внутри процессора, внутри памяти.

Команда пересылки данных должна содержать следующую информацию:

– адреса источника и получателя операндов – адреса ячеек памяти, номера регистров ЦП или информацию о том, что операнды – в стеке;

– длина подлежащих пересылке данных (в байтах или словах), заданная явно либо косвенно;

– способ адресации каждого из операндов (т. е. каким образом должен вычисляться физический адрес операнда).

Команды арифметической и логической обработки

Обеспечивают арифметическую и логическую обработку информации в различных формах ее представления.

Помимо вычисления результата, формируются признаки (флаги) в АЛУ (арифметико-логическом устройстве), характеризующие результат (например, Zero, Negative, Overflow, Carry).

Библиотека БГУИР

Подробнее этот тип команд будет рассмотрен в лабораторной работе, связанной с АЛУ.

Команды SIMD – Single Instruction Multiple Data

Эти команды обрабатывают сразу две группы чисел. Операнды обычно представлены в упакованном формате.

Команды работы со строками

Обычно предусматриваются команды перемещения, сравнения и поиска строк. В большинстве ВМ данные операции имитируются с помощью других команд.

Команды преобразования

Изменение формата представления данных. Например, перевод из двоичной системы счисления в двоично-десятичную.

Команды ввода/вывода

Могут быть разделены на 3 типа:

– команды управления периферийными устройствами (ПУ);

– команды проверки состояния ПУ;

– команды ввода /вывода.

Команды управления системой

Эти команды являются привилегированными командами и выполняются, когда ЦП находится в привилегированном состоянии либо выполняет привилегированную программу (обычно ОС).

Только с помощью данных команд может быть изменено состояние ряда регистров устройства управления.

Команды управления потоком команд

Вместо инкрементации счётчика команд команда загружает в счётчик адрес следующей команды.

Существует три типа таких команд:

– безусловные переходы;

– условные переходы (ветвления);

– вызовы процедур и возврат из процедур.

Для команд статического перехода один из вариантов – указание адреса перехода через счетчик команд (PC – *program counter* или IP – *instruction pointer*). В случае динамического управления должен быть иной способ указания адреса перехода.

Команды вызова процедуры обеспечивают переход из текущей точки программы к начальной точке процедуры. Команды возврата из процедуры обеспечивают возврат к команде, следующей за командой вызова. Команда возврата должна обеспечиваться средствами сохранения текущего состояния счётчика команд и его восстановления.

Библиотека БГУИР

4.1.2. Форматы команд

В команде для сведений о типе операции, адресной информации о нахождении обрабатываемых данных, а также для информации о месте хранения результатов выделяются определенные разряды (поля). В общем случае команду можно разбить на 2 части: операционную и адресную.

Операционная часть	Адресная часть

Типовая команда должна определять:

– подлежащую выполнению операцию;

– адреса исходных данных, над которыми выполняется операция;

– адрес, по которому должен быть помещён результат операции.

Выбор формата команд при создании ВМ влияет на многие характеристики будущей машины.

Необходимо принимать во внимание также следующие факторы:

– общее число различных команд;

– общую длину команды;

– тип полей команды (фиксированной или переменной длины) и их длину;

– простоту декодирования;

– адресуемость и способы адресации;

– стоимость оборудования для декодирования и исполнения команды.

Длина команды влияет на организацию и ёмкость памяти, структуру шин, сложность и быстродействие ЦП.

Чем длиннее команда, тем медленнее её выборка. С целью оптимизации пересылки данных по шине длина команды обычно кратна разрядности шины данных. Если какие-то биты команды не используются, их обычно резервируют для будущих поколений процессоров разрабатываемого семейства.

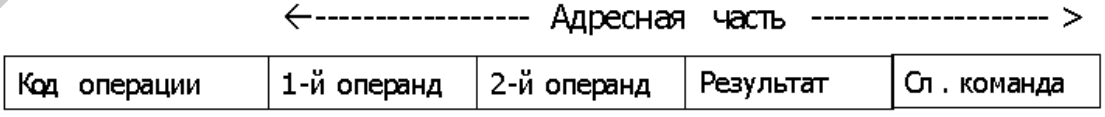
Различные способы адресации приводят к общему виду команды:

КОП – Код операции	СА – Способ адресации	Адресная часть

В большинстве ВМ одновременно используются несколько различных форматов команд.

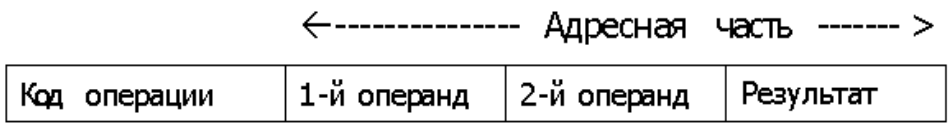
Классификация по количеству адресов в команде:

– четырёхадресный формат:

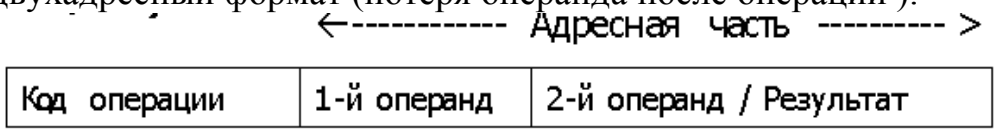


Библиотека БГУИР

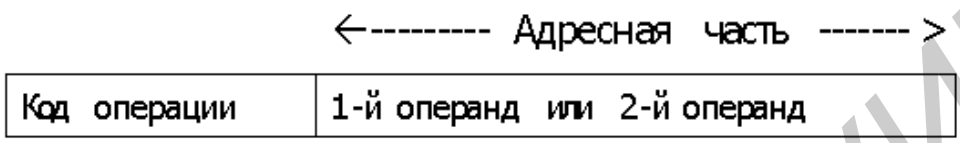
– трёхадресный формат – добавляются команды переходов:



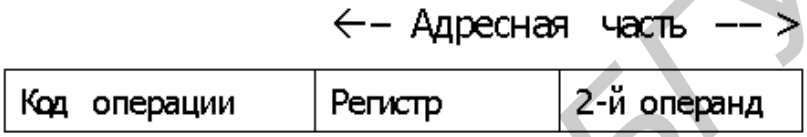
– двухадресный формат (потеря операнда после операции):



– одноадресный формат (аккумуляторная архитектура):



– полутораадресный или регистровый формат:



– нуль-адресный формат (отдельные команды):

Критерии выбора адресности команд:

– ёмкость запоминающего устройства;

– время выполнения программы;

– эффективность использования ячеек памяти при хранении программы.

С точки зрения увеличения ёмкости и эффективности использования памяти наиболее оптимальным вариантом будут одноадресные команды.

Суммарные потери времени для трёхадресной команды складываются из:

– времени на выборку команды;

– времени на выборку первого операнда;

– времени на выборку второго операнда;

– времени на запись результата в память.

Для одноадресной команды составляющие потерь по времени следующие:

– время на выборку команды;

– время на выборку операнда.

Определяющим фактором являются типы алгоритмов, на которые ориентирована ВМ. В последовательных программах результат предыдущей операции используется для следующей, и выгодна одноадресная команда. В параллельных программах результат пересылается в память, и трёхадресные команды будут эффективнее. В комбинированных программах лучше использовать одноадресные и полутораадресные команды.

Библиотека БГУИР

Оттранслированные команды записываются в соседние ячейки памяти (памяти программ, если память программ и данных разделены) в порядке их следования в программе. При естественном порядке выполнения команд в программе адрес каждой следующей команды определяется по содержимому специального счетчика команд (IP или PC), который входит в состав процессора. Содержимое этого счетчика автоматически наращивается на 1 при выполнении очередной команды. При организации ветвления цикла или для перехода на подпрограмму в счетчик команд принудительно записывается адрес перехода, указанный в ходе команды.

4.1.3. Способ адресации операндов

Исполнительный адрес операнда ($А_{исп}$) – двоичный код номера ячейки памяти, служащей источником /приёмником операнда (адрес на ША или номер регистра).

Адресный код команды ($А_к$) – двоичный код в адресном поле команды, из которого необходимо сформировать исполнительный адрес операнда.

В современных ВМ, как правило, $А_{исп}$ и $А_к$ не совпадают, требуется соответствующее преобразование. Поэтому способ адресации можно определить как способ формирования исполнительного адреса по адресному коду команды.

Существуют 2 различных принципа поиска операндов в памяти: **ассоциативный** и **адресный**.

Ассоциативный поиск (поиск по содержанию запоминающей ячейки) предполагает просмотр содержимого всех ячеек памяти для выявления кода, содержащего заданный командой ассоциативный признак.

Адресный поиск предполагает, что операнд находится по адресу, указанному в адресном поле команд.

Способы адресации операндов можно разбить на несколько групп:

1) по наличию адресной информации в команде (явная и неявная);
2) по кратности обращения в оперативную память;
3) по способу формирования адресов ячеек памяти.

По способу формирования адресов ячеек памяти различают:

– адресацию с абсолютным способом формирования адресов ячеек памяти (двоичный код адреса ячейки памяти может быть извлечен целиком либо из адресного поля команды, либо из какой-нибудь другой ячейки памяти);

– адресацию с относительным способом формирования адресов ячеек памяти (двоичный код адресной ячейки памяти образуется из нескольких составляющих).

По кратности обращения в оперативную память различают:

– непосредственную адресацию *(direct addressing);*
– прямую адресацию *(immediate addressing);*
– косвенную адресацию *(indirect addressing).*

Рассмотрим эти способы адресации и их различные варианты более подробно.

Библиотека БГУИР

Непосредственная адресация (Immediate Addressing)

При непосредственной адресации операнд располагается в адресном поле команды. Используется в операциях сравнения, загрузки констант в регистры, арифметических операциях.

КОП	Операнд

Пример: mov AX, 300

Прямая адресация (Direct Addressing)

При прямой адресации обращение к операнду производится по адресному коду в поле команды. При этом исполнительный адрес совпадает с адресным кодом команды (рис. 4.1).

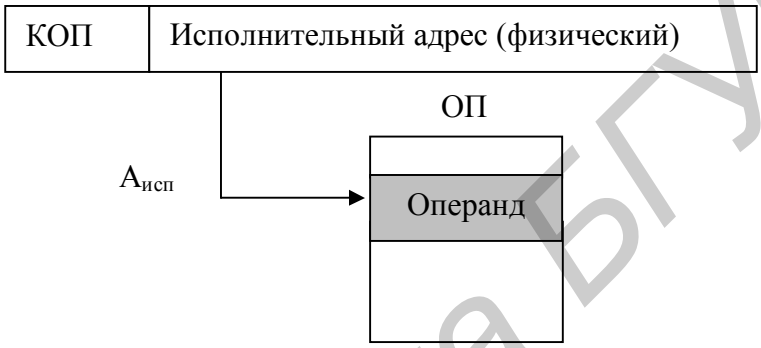


Рис. 4.1. Прямая адресация

Косвенная адресация (Indirect Addressing)

При косвенной адресации код команды указывает адрес ячейки памяти, в которой находится не сам операнд, а его исполнительный адрес.

Косвенная адресация памяти может быть многоуровневой с глубиной 2 и более, но с увеличением глубины после 10 увеличивается количество обращений за следующими адресами памяти. Косвенная адресация используется для организации цепных списков.

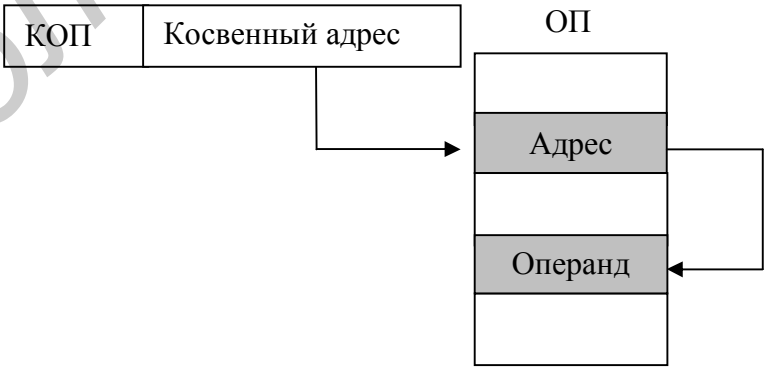


Рис. 4.2. Косвенная адресация

Плюсом этого способа является возможность изменения адреса операнда в процессе выполнения программы.

Библиотека БГУИР

Прямая регистровая адресация (Register Direct Addressing)

Кроме оперативной памяти (ОП), операнды могут находиться в регистрах общего назначения (РОН). При прямой регистровой адресации в регистрах содержится сам операнд (операнды), что иллюстрирует рис. 4.3.

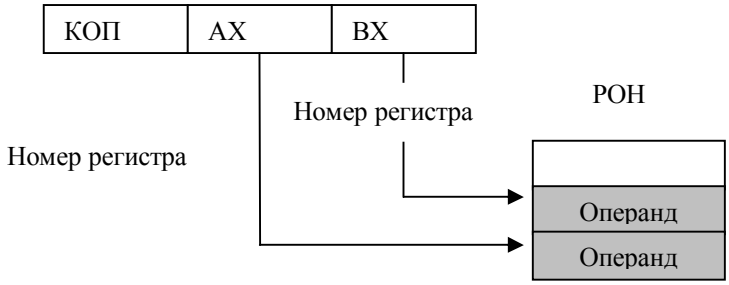


Рис. 4.3. Прямая регистровая адресация

Плюс прямой регистровой адресации – небольшая длина адресной части при условии небольшого числа РОНов.

Косвенная регистровая адресация (Register Indirect Addressing)

При косвенной регистровой адресации (рис. 4.4) в регистрах содержится адрес операнда (операндов).

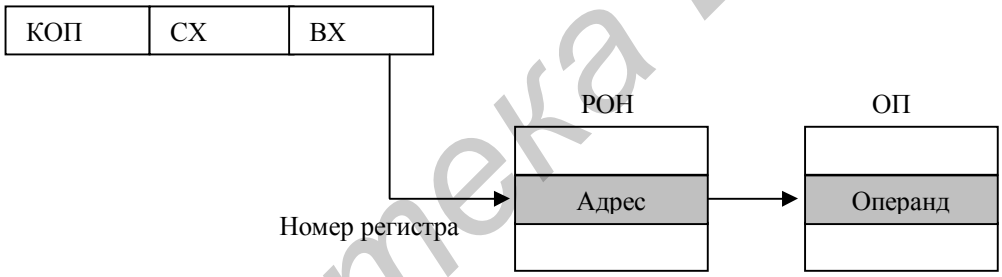


Рис. 4.4. Регистровая косвенная адресация

Адресация с относительным способом формирования адресов ячеек памяти предполагает, что двоичный код адресной ячейки памяти (исполнительный адрес) образуется из нескольких составляющих:

- базы;
- индекса;
- смещения.

$$A_{исп} = [\text{Базовый регистр}] + [\text{Индексный регистр}] + \text{Смещение}$$

Эти составляющие используются в различных сочетаниях:

- $A_{исп}$ = [Базовый регистр] + Смещение – базовая адресация со смещением (рис. 4.5);
- $A_{исп}$ = [Индексный регистр] + Смещение – индексная адресация со смещением (рис. 4.6);
- $A_{исп}$ = [BX] + [SI] + Смещение – базово-индексная адресация со смещением (рис. 4.7).

Библиотека БГУИР

– автоинкрементная индексная адресация: когда адрес, находящийся в индексном регистре, автоматически увеличивается (автоинкрементная адресация) или уменьшается (автодекрементная адресация) на постоянную величину до или после выполнения операции;

– поставтоинкрементная адресация: индексный регистр увеличивается после выполнения команды;

– предавтоинкрементная адресация: индексный регистр увеличивается до выполнения команды.

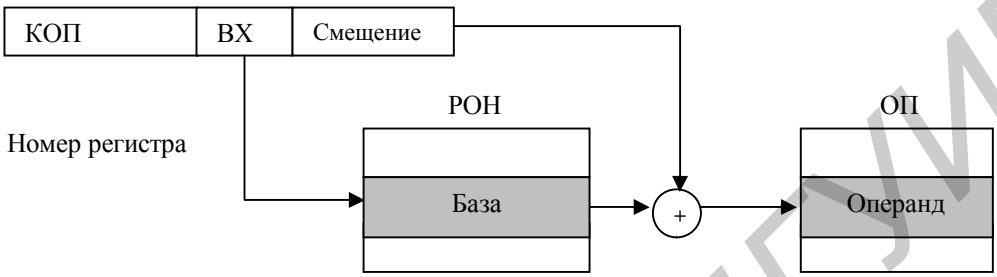


Рис. 4.5. Базовая адресация со смещением

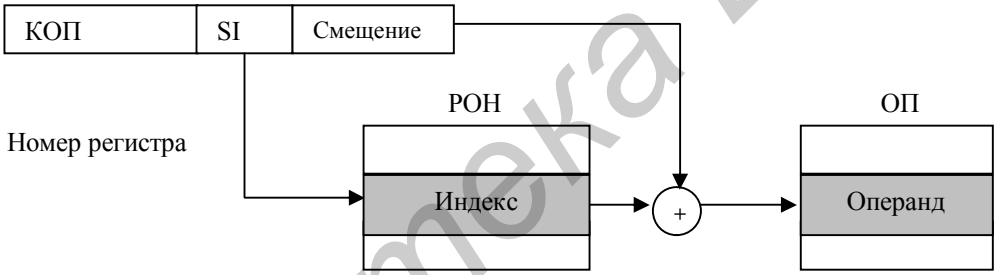


Рис. 4.6. Индексная адресация со смещением

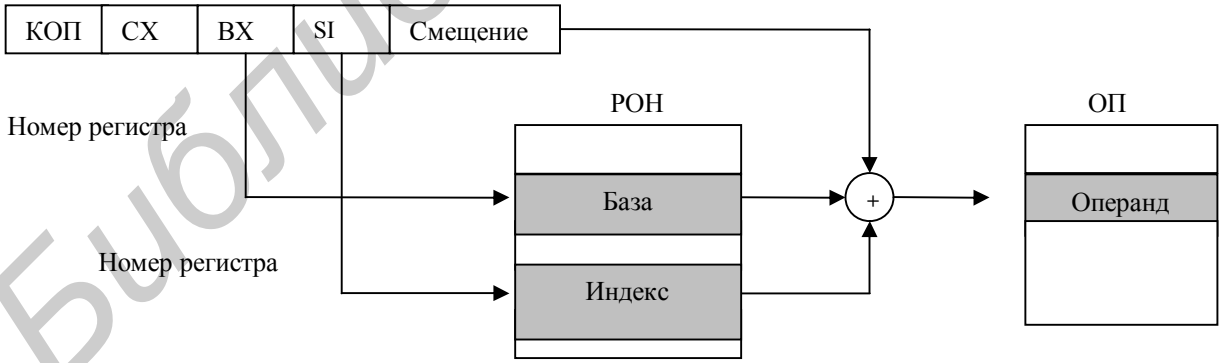


Рис. 4.7. Базово-индексная адресация со смещением

Адресация со смещением (Displacement)

Исполнительный адрес формируется суммированием адресного поля команды и содержимого одного или нескольких РОНов или специализированных регистров (базового или индексного), рис. 4.5. – 4.7.

Библиотека БГУИР

Относительная адресация со смещением

Для получения исполнительного адреса содержимое адресного поля складывается со счетчиком команд (IP или PC), что показано на рис. 4.8.

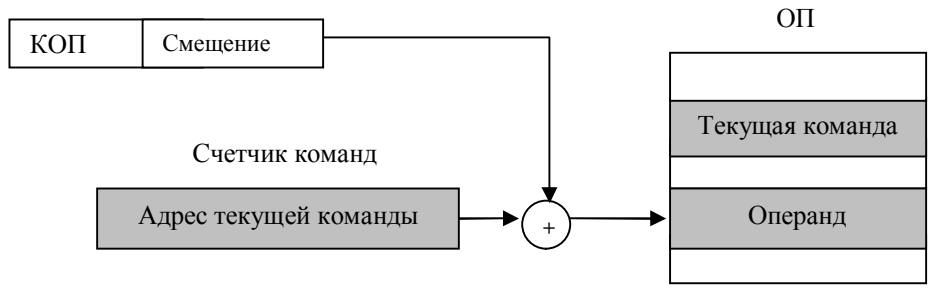


Рис. 4.8. Относительная адресация со смещением

Неявная адресация

При неявной адресации адресное поле в команде отсутствует, а адрес операнда определяется кодом операции. Например, из команды может быть исключен адрес приемника адресата, при этом подразумевается, что результат записывается на месте второго операнда.

Примеры: MUL BX
DIV BX

Блочная адресация

Используется в командах, обрабатывающих блок данных. Блок обычно описывается адресом первой или последней ячейки и количеством элементов в блоке. Иногда в качестве конца блока удобно использовать некоторый признак.

Стековая адресация

Для чтения записи доступен только один регистр – вершина стека. Этот способ адресации используется, в частности, системой прерывания программ при вложенных вызовах подпрограмм.

Стековая память реализуется на основе обычной памяти с использованием указателя стека и автоиндексной адресации. Запись в стек производится с использованием автодекрементной адресации, а чтение – с использованием автоинкрементной адресации.

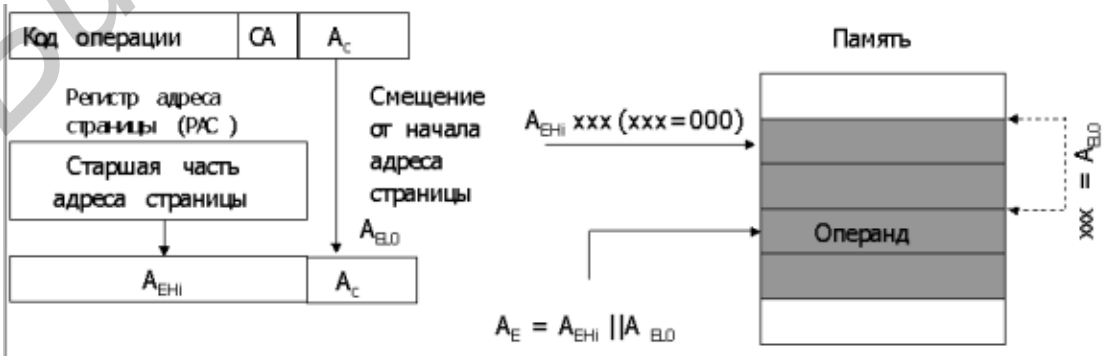


Рис. 4.9. Страничная адресация

Библиотека БГУИР

Страничная адресация

При организации названной адресации проводится разбиение адресного пространства на страницы (рис. 4.9). Начальный адрес страницы является базой. При этом старшая часть адреса страницы хранится в специальном регистре (РАС). Адресный код, таким образом, задаёт смещение внутри страницы.

4.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

В процессе выполнения лабораторной работы можно использовать наработки с предыдущих лабораторных работ. На выходе должна получиться система, состоящая из блока памяти, блока управления, блока РОНов и шин данных, адреса и управления.

Для выполнения задания необходимо выполнить следующие этапы:

4.2.1. Ввести шину данных (ШД), шину адреса (ША) и шину управления (ШУ).

4.2.2. Разделить память на память данных (блок RAM) и память команд (блок ROM). На адресные входы памяти завести ША. Ввод и вывод данных осуществлять через ШД. Объединить два типа памяти в один блок.

4.2.3. Ввести блок управления, в котором производить дешифрацию команд и выдавать управляющие сигналы и сигналы синхронизации памяти и регистров в зависимости от выполняемой команды.

4.2.4. Ввести блок регистров общего назначения (РОН) и управляющую логику для него.

4.2.5. Ввести специальные регистры, разрядность которых определяется разрядностью шины данных. Физически разместить их в блоке управления.

Специальные регистры:

1. **Счетчик команды IP** (Instruction Pointer), который служит для фиксации и формирования адресов команд, располагающихся в ячейках памяти. Необходимо обеспечить автоинкремент этого регистра.

2. **Регистр команд IR** (Instruction Register), который предназначен для хранения кода команды или его части в процессе выполнения команды после ее приема из памяти.

3. **Регистр адреса операнда AR** (Address Register), в который записывается исполнительный адрес операнда.

4. **Регистр данных DR** (Data Register), в который помещаются непосредственно сами операнды.

5. **Регистр признаков результата или регистр флагов FR** (Flag Register), в котором необходимо задать биты для переполнения, признака нуля, признака знака, переноса.

Интерфейс с внешней средой у верхнего блока иерархии должен ограничиваться стартстопным сигналом и сигналом внешнего тактирования clk.

Выполняемую команду можно разбить на несколько этапов. Для разделения этих этапов во времени можно использовать блок опорных сигналов **DCa**

Библиотека БГУИР

или блок генерации сигнала специальной формы, разработанные в лабораторной работе №2.

Необходимо выполнить цикл команд, заданных по варианту. Цикл состоит из следующих фаз:

– выборки команды и формирования адреса следующей команды;
– декодирования команды;
– формирования исполнительных адресов операндов;
– выборки операндов;
– исполнения операции;
– записи результата.

Некоторые шаги цикла команд могут отсутствовать, это зависит от выполняемой команды.

Если при анализе команды вы приходите к выводу, что некоторые специальные регистры не востребованы ввиду отсутствия функционала для этого регистра, то в операции вы их можете не использовать, но сама их реализация должна присутствовать.

Семейство ПЛИС для реализации – Flex10K (изменить семейство ПЛИС *Family* можно в настройках проекта в ветке *Device* или в пункте меню *Assignments→Device)*.

Таблица 4.1

Варианты заданий к лабораторной работе №4

№	Разр. ША	Разр. ШД	Команда 1	Адресация операндов команды 1	Команда 2	Адресация операндов команды 2
1	8	8	R→M	Прямая регистровая	JMP adr	Непосредственная
2	4	8	M→R	Прямая	JZ adr	Относительная со смещением
3	8	4	const→R	Непосредственная	JS adr	Относительная со смещением
4	4	4	const→M	Непосредственная	JCadr	Непосредственная
5	8	8	M→R	Косвенная	JMP adr	Относительная со смещением
6	4	8	R→M	Неявная	JZ adr	Непосредственная
7	8	4	R+C→M	Со смещением	JS adr	Непосредственная
8	4	4	R→M	Косвенная регистровая	JCadr	Относительная со смещением

Библиотека БГУИР

4.3. СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

1. Задание.
2. Проектирование блока управления выборкой команд.
   2.1. Схема блока, самого верхнего в иерархии проекта.
   2.2. Схемы входящих в него блоков.
3. Результаты моделирования общей схемы.

ЛАБОРАТОРНАЯ РАБОТА №5
Арифметико-логическое устройство

5.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

5.1.1. Команды арифметической и логической обработки

Команды арифметической и логической обработки обеспечивают арифметическую и логическую обработку информации в различных формах ее представления.

Для каждой формы представления чисел в архитектуре системы команд предусматривается определённый стандартный набор операций.

Помимо вычисления результата, формируются признаки (флаги) в АЛУ, характеризующие результат (например – Zero, Negative, Overflow, Carry).

Стандартный набор операций с целыми числами:

– двухместные (с двумя операндами) арифметические операции: «+», «–», «*», «/»;

– одноместные арифметические операции. Например, абсолютное значение, изменение знака операнда;

– операции сравнения, устанавливающие признаки, характеризующие соотношения между величинами (=, <>, <, >, <=, >=).

Список часто дополняют такие операции, как вычисление остатка от целочисленного деления, сложение с учётом переноса, вычитание с учётом займа, инкремент, декремент.

Типичный набор операций с числами в форме с плавающей запятой (ПЗ):

– основные арифметические операции: «+», «-», «*», «/»;

– операции сравнения, устанавливающие признаки, характеризующие соотношения между величинами (=, <>, <, >, <=, >=);

– операции преобразования: формы представления (между формой с фиксированной запятой (ФЗ) и ПЗ, формата представления (с одинарной и двойной точностью).

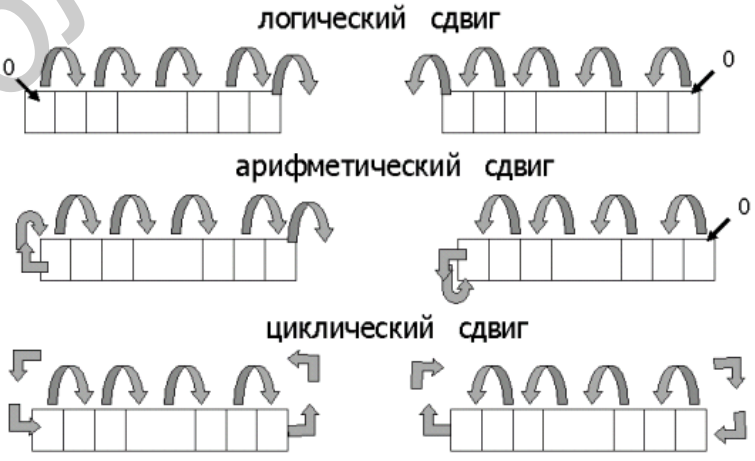


Рис. 5.1. Типы сдвиговых операций

Библиотека БГУИР

Логические команды предназначены для выполнения логических операций как над отдельными битами слов, так и целыми байтами, словами и т. п.

Минимальный набор – «НЕ», «И», «ИЛИ », «сложение по модулю 2».

В дополнение практически во всех архитектурах системы команд (АСК) предусмотрены команды логического, арифметического и циклического сдвигов (рис. 5.1).

Логический сдвиг – это сдвиг, при котором уходящий бит уходит, не влияя на оставшееся биты, а на месте появившегося бита записывается бит 0. Пример работы операции сдвига: число 10101010b дает число 01010100b, если сделать сдвиг влево на 1 бит, и число 01010101b, если сделать сдвиг вправо на 1 бит.

Арифметический сдвиг. При этом виде сдвига слово рассматривается не просто как группа битов, а как целое число в дополнительном коде. При сдвиге влево происходит логический сдвиг, при сдвиге вправо уходящий бит уходит, не влияя на оставшиеся биты, а на месте появившегося бита устанавливается бит, соответствующий знаку. Пример работы операции сдвига: число 11111010b = минус 6 дает число 11110100b = минус 12, если сделать сдвиг влево на 1 бит, и число 11111101b = минус 3, если сделать сдвиг вправо на 1 бит.

Легко заметить, что при арифметическом сдвиге сдвиг влево соответствует умножению на 2, а сдвиг вправо – делению на 2.

Циклический сдвиг. При этом виде сдвига уходящий бит появляется на месте появляющегося. Пример работы операции сдвига: число 11111010b дает число 11110101b, если сделать сдвиг влево на 1 бит, и число 01111101b, если сделать сдвиг вправо на 1 бит.

Операции с десятичными числами:

Ранее широко использовались специальные форматы хранения – двоично-десятичный код (BCD – *Binary Coded Decimal*), которые поддерживали два формата представления чисел: зонный и упакованный. В силу особенностей упакованного формата умножение и деление было возможно только в зонном формате.

В ВМ первых поколений в АСК были специальные команды («+», «–», «*», «/») для десятичных чисел. В АСК современных машин соответствующие вычисления имитируются с помощью команд целочисленной арифметики.

5.1.2. Разработка АЛУ

Для разработки блока АЛУ рекомендуется следующая последовательность действий:

1) разработать непосредственно блоки, выполняющие каждую из заданных по варианту операций;

2) реализовать выбор операции в АЛУ в зависимости от управляющих сигналов (на формирование которых влияет дешифрованная команда), причем при реализации функций использовать только логические операции;

3) определить источники операндов, а также приемники результата;

Библиотека БГУИР

4) разработать блок управления АЛУ;

5) проверить работу АЛУ как отдельного модуля, сформировав управляющие сигналы самостоятельно;

6) проверить работу всего проекта, предварительно задав в памяти команд команды для работы с АЛУ.

В качестве примера приведено АЛУ, реализованное на одноадресном регистровом ЗУ и комбинационной логике (рис. 5.2).

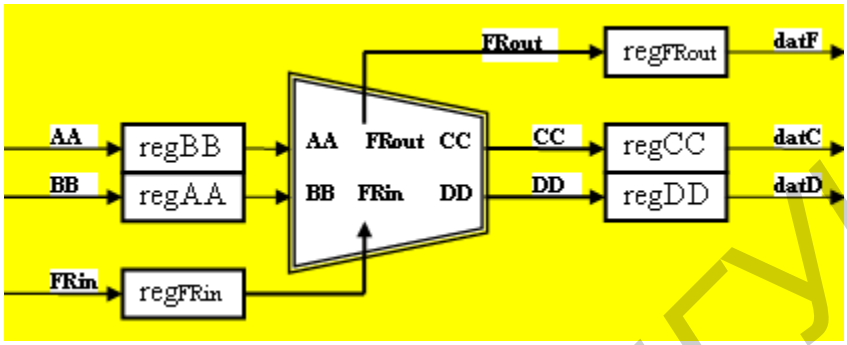


Рис. 5.2. АЛУ на одноадресном ЗУ и комбинационной логике

В одноадресном ЗУ невозможно одновременное обращение по двум адресам и считывание/запись двух данных. Для временного хранения двух операндов в схеме установлены буферные регистры **regAA**и **regBB**, в которые через мультиплексор поочередно загружаются операнды A и B с шины данных, куда их «выбрасывает» АЛУ-команда. Некоторые АЛУ-команды используют признаки-флаги предыдущих команд и/или вырабатывают свои признаки (флаги) результатов: выходные флаги заносятся в **regFRout**, входные флаги поступают из **regFRin**, куда дублируются из регистра флагов **FR** по шине данных. Через некоторое время на выходе появляются АЛУ-результаты: **CC, DD**и **FRout,** которые фиксируются в выходных регистрах **regCC, regDD**и**regFRout.** Отсюда по шине данных они переносятся в блок РОНов: **regCC =>RON#0, regDD=> =>RON#1 –** соответственно и **regFRou =>regFR** блока регистра флагов УУ. Последовательность управляющих сигналов, реализующих перечисленные действия для команд NOT, AND, генерируется блоком управления АЛУ в главном блоке управления.

5.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

При выполнении лабораторной работы необходимо использовать наработки из предыдущей лабораторной работы.

К архитектуре системы из предыдущей лабораторной работы необходимо добавить блок АЛУ и управление для него.

Для этого следует реализовать полный цикл выполнения заданных по варианту команд АЛУ. Напомним, что цикл команды состоит из следующих фаз:

- выборки команды и формирования адреса следующей команды;
- декодирования команды;

Библиотека БГУИР

- формирования исполнительных адресов операндов;
- выборки операндов;
- исполнения операции;
- записи результата.

Ненужные для заданной команды фазы отбрасываются. В процессе выполнения фаз должны по назначению использоваться специальные регистры (IP, IR, AR, DR, FR).

Арифметические команды необходимо выполнить на логических элементах.

Семейство ПЛИС для реализации – Flex10K (изменить семейство ПЛИС *Family* можно в настройках проекта в ветке *Device* или в пункте меню *Assignments→Device)*.

Таблица 5.1

Варианты заданий к лабораторной работе №5

№ Варианта	**Команда сдвига**	**Арифметическая команда, адресация операндов**	**Логическая команда**
1	SLL (Shift Left Logical)	INCS, непосредственная	NOT
2	SRL (Shift Right Logical)	ADDC, прямая регистровая	AND
3	SLA (Shift Right Arithmetical)	SUB, косвенная регистровая	OR
4	SRA (Shift Right Arithmetical)	CMP, прямая регистровая	NOR
5	ROL (Rotate Left Logical)	INCS, прямая	NAND
6	ROL (Rotate Right Logical)	ADDC, косвенная регистровая	NOTZ
7	SLA (Shift Right Arithmetical)	SUB, прямая регистровая	XOR
8	SRA (Shift Right Arithmetical)	CMP, прямая	NXOR

5.3. СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

1. Задание.
2. Проектирование исполнительных модулей процессора.
   - 2.1. Схема блока, самого верхнего в иерархии проекта.
   - 2.2. Схема блока выборки команд из лабораторной работы №4.
   - 2.3. Схемы блока АЛУ и входящих в него блоков.
   - 2.4. Схема блока управления АЛУ.
3. Результаты моделирования.

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №6
Стековое запоминающее устройство

6.1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Доступ к стековой памяти происходит в большинстве процессоров в следующих ситуациях:

1) при выполнении непосредственных команд стекового доступа: «поместить в стек» – push или «извлечь из стека» – pop;

2) при выполнении команды вызова подпрограммы и при возврате из нее;

3) при входе в прерывание и возврате из него.

Положение стека в адресуемой памяти определяется содержимым указателя стека (SP – *stack pointer*).

Указатель стека программно доступен, программист может задать его значение обычной командой пересылки: *mov sp, #1000h* – задает положение стека, начиная с адреса 1000h.

Содержимое указателя стека используется как адрес операнда-приемника при записи или как адрес операнда-источника при считывании из стека.

При обращении к стеку автоматически модифицируется и содержимое указателя стека, чтобы обеспечить следующее обращение к очередной ячейке стека. Обращение к стеку – косвенно-регистровая адресация через указатель стека с автоиндексацией.

Организация стека зависит от следующих факторов.

1. Направление роста стека. Если при записи в стек содержимое указателя стека автоматически увеличивается (и, соответственно, при считывании автоматически уменьшается), то говорят, что стек растет в сторону увеличения адресов. В противоположном случае говорят, что стек растет в сторону уменьшения адресов.

2. Если модификация указателя стека выполняется до записи и соответственно после считывания, то указатель стека всегда указывает на последнюю занятую ячейку стека. Наоборот, если модификация производится после записи и до считывания, указатель стека всегда указывает на первую свободную ячейку стека.

6.2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

К архитектуре системы из предыдущей лабораторной работы необходимо добавить блок стекового ЗУ и управление для него.

Реализовать стековое ЗУ на регистрах, количество которых зависит от объема стека, заданного по варианту. При этом должен присутствовать порт, показывающий переполнение стека.

В таблице вариантов указано направление роста стека и место памяти стека, на которое указывает указатель стека.

Ввести в систему команд две команды для стека: *push* и *pop* для помещения и извлечения данных из стека соответственно. В качестве источника и при-

Библиотека БГУИР

емника для этих команд использовать регистры общего назначения. Продемонстрировать работу этих команд.

Реализовать занесение результатов арифметической операции по варианту из предыдущей лабораторной работы в стек.

Семейство ПЛИС для реализации – Flex10K (изменить семейство ПЛИС *Family* можно в настройках проекта в ветке *Device* или в пункте меню *Assignments→Device)*.

Таблица 6.1

Варианты заданий к лабораторной работе №6

№ варианта	**Объем стека**	**Направление роста стека**	**Указатель стека указывает на**
1	4 байта	Вверх	первую свободную ячейку
2	5 байт	Вниз	последнюю занятую ячейку
3	6 байт	Вниз	первую свободную ячейку
4	7 байт	Вверх	последнюю занятую ячейку
5	8 байт	Вверх	первую свободную ячейку
6	9 байт	Вниз	последнюю занятую ячейку
7	10 байт	Вниз	первую свободную ячейку
8	11 байт	Вверх	последнюю занятую ячейку

6.3. СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

1. Задание.
2. Проектирование исполнительных модулей процессора.
   2.1. Схема блока выборки команд из лабораторной работы №4.
   2.2. Схема самого верхнего блока в иерархии проекта.
   2.3. Схемы блока стекового ЗУ.
   2.4. Схема блока управления стеком.
3. Результаты моделирования блока стекового ЗУ в общей системе.

Библиотека БГУИР

Литература

1. Столлингс, У. Структурная организация и архитектура компьютерных систем / У. Столлингс; пер. с англ. – 5-е изд. – М. : Вильямс, 2001. – 892 с.

2. Таненбаум, Э. Архитектура компьютерных систем / Э. Таненбаум ; пер. с англ. – 4-е изд. – М. : Питер, 2002. – 698 с.

3. Цилькер, Б. Я. Организация ЭВМ и систем / Б. Я. Цилькер, С. А. Орлов. – М. : Питер, 2007. – 668 с.

4. Грушвицкий, Р. Проектирование систем на микросхемах программируемой логики / Р. Грушвицкий. – СПб. : Питер, 2002. – 608 с.

Библиотека БГУИР

Св. план 2009, поз. 137

Учебное издание

Самаль Дмитрий Иванович
Колонов Виталий Валентинович

СТРУКТУРНАЯ И ФУНКЦИОНАЛЬНАЯ ОРГАНИЗАЦИЯ ЭВМ. ОСНОВНЫЕ БЛОКИ

Лабораторный практикум
для студентов специальности
1-40 02 01
«Вычислительные машины, системы и сети»
всех форм обучения

Библиотека БГУИР

Редактор Г. С. Корбут
Корректор Е. Н. Батурчик
Компьютерная верстка А. В. Бас

Подписано в печать 17.11.2011.	Формат 60x84 1/16.	Бумага офсетная.
Гарнитура «Таймс».	Отпечатано на ризографе .	Усл. печ. л. 2,67.
Уч.-изд. л. 3,0.	Тираж 100 экз.	Заказ 546.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6