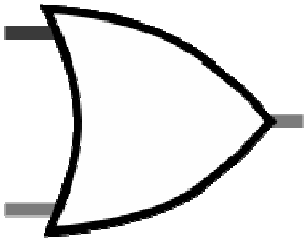
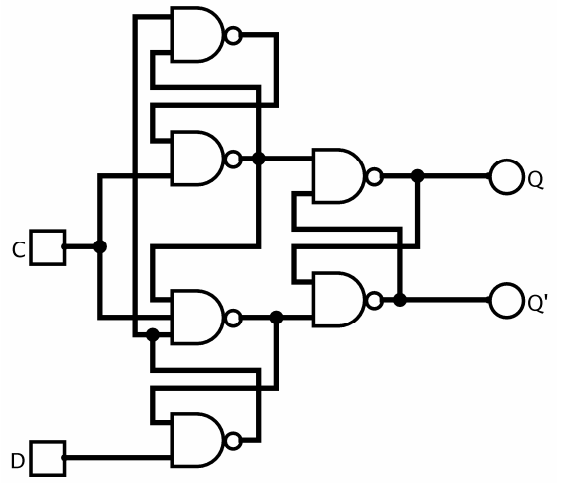
**МГУП**

Попов Д.И., Лилов И.П.

# Организация ЭВМ

## Лабораторные работы в программе Logisim






Москва 2011

# Оглавление

Введение ..... 5
1. Основы работы с цифровыми сигналами ..... 7
1.1. Логические 0 и 1 ..... 7
1.2. Булевы функции и логические элементы ..... 8
1.3. Комбинационные и последовательностные логические устройства 13
1.4. Минимизация булевых функций ..... 14
1.5. Задержки устройств ..... 19
1.6. Тактовые импульсы. Синхронные и асинхронные устройства ..... 20
2. Программа Logisim ..... 24
2.1. Знакомство с программой ..... 24
2.2. Подсхемы ..... 25
2.3. Провода ..... 26
2.4. Комбинационный анализ ..... 27
2.5. Библиотека компонентов ..... 28
2.6. Различные возможности ..... 31
3. Лабораторные работы ..... 33
3.1. Лабораторная работа 1. Знакомство с программой Logisim. Комбинационные устройства. Минимизация ..... 33
3.2. Лабораторная работа 2. Элементарные устройства памяти ..... 37
3.3. Лабораторная работа 3. Декодер, мультиплексор и демультиплексор ..... 42
3.4. Лабораторная работа 4. Арифметика. АЛУ ..... 48
3.5. Лабораторная работа 5. Тактовые импульсы. Шина ..... 63
3.6. Лабораторная работа 6. Сложные устройства памяти: ОЗУ и ПЗУ 69
3.7. Лабораторная работа 7. Устройства ввода и вывода ..... 73
3.8. Лабораторная работа 8. Ассемблер. ..... 80
4. Курсовой проект ..... 99
4.1. Общее описание проекта и требования ..... 99
4.2. Варианты устройств для реализации ..... 100
4.2.1. Система управления светофорами ..... 100
4.2.2. Банкомат ..... 100
4.2.3. Автомат выдачи ..... 101
4.2.4. Система управления лифтами ..... 101
4.2.5. Дозиметр ..... 101
4.2.6. Бортовая система автомобиля ..... 101
4.2.7. Часы-будильник ..... 102

4.2.8. Микроволновая печь ..... 102
4.2.9. Стиральная машина ..... 102
4.2.10. Кондиционер ..... 102
4.2.11. Велоспидометр ..... 103
4.2.12. Машина Поста ..... 103
4.2.13. Клеточные автоматы ..... 103
4.2.14. Бегущая строка ..... 104
4.2.15. Игры ..... 104
4.2.16. Робот ..... 104
4.2.17. Калькулятор ..... 105
4.2.18. Процессор ..... 105

# Введение

Знание подробностей принципов работы цифровых устройств отнюдь не является необходимым для работы программиста, дизайнера, или любого другого специалиста, использующего компьютер. Однако более глубокое понимание внутреннего устройства вычислительной техники (и уж тем более — умение её проектировать) во многих случаях может повысить эффективность работы специалиста; стать основой для идей и решений, которые никогда бы не возникли без этого понимания. Кроме того, эти знания способны дать то удивительное чувство, которое возникает, когда вещь, используемая для повседневной работы (в нашем случае — компьютер), больше не выглядит загадочным «чёрным ящиком»; когда процессы, происходящие в ней на всех уровнях, представляются понятными и логичными.

Подразумевается, что данное пособие держит в руках человек, с одной стороны изучивший курсы электротехники и электроники, а с другой — знакомый с программированием и математической логикой. Задача данного курса лабораторных работ — заполнить пробел между этими областями прикладного знания; показать стройную взаимосвязь между различными уровнями абстракции, используемыми в цифровой технике. Таким образом, читатель сделает ещё один шаг от разрозненных знаний о мире (в нашем случае об одной его маленькой части — цифровой электронике) к целостной картине. Цель, более чем стоящая прикладываемых усилий.

Данное пособие содержит краткое изложение теоретических сведений, необходимых для выполнения лабораторных работ; описание самих лабораторных работ, и подробное задание для курсового проекта.

Теоретическая часть отнюдь не претендует на полноту, это скорее необходимый минимум; однако эти сведения могут служить отправной точкой для отыскания более полной информации (разумеется, при наличии необходимости и желания).

Сами лабораторные работы выполняются в замечательном образовательном инструменте Logisim. Особо отметим, что несмотря на все его достоинства, основной целью данного курса является изучение внутреннего устройства цифровой техники, а не изучение конкретной программы. Именно поэтому нюансы работы с Logisim в большей степени, чем другие составляющие, оставлены на самостоятельное изучение. Это не станет проблемой, так как с Logisim распространяется

подробнейшая справочная система на русском языке. Некоторые из лабораторных работ не имеют заданий по вариантам. Это означает, что контроль выполнения работ и усваивания материала в большей степени осуществляется преподавателем. При защите работ вопросы по работе схемы, небольшие дополнительные задания по изменению схемы и т.п. просто необходимы.

Логическим завершением курса является курсовой проект. Студент должен в индивидуальном порядке спроектировать несложное, практически применимое устройство. В данном пособии приводится множество вариантов таких устройств с описаниями и некоторыми идеями по реализации. Однако приветствуются любые собственные идеи студентов — от внесения изменений в задание до полностью оригинальных проектов. Разумеется, по согласованию с преподавателем. Отметим, что данный курс предполагает очень активный диалог преподавателя и студентов.

# 1. Основы работы с цифровыми сигналами

## 1.1. Логические 0 и 1

В цифровой технике любая информация представляется в виде набора цифровых сигналов. *Цифровой сигнал* — это дискретный сигнал, квантованный по амплитуде. Например, в устройстве имеется контакт, сигнал на котором нас интересует. В каждый момент времени электрический потенциал на этом контакте относительно выбранного нулевого потенциала будет иметь одно из конечного числа значений. На протяжении всего курса мы будем иметь дело только с *двоичной логикой*, то есть потенциал может иметь лишь два разных значения: логический 0 (в таком случае говорят о *сигнале низкого уровня*) или логическая 1 (*сигнал высокого уровня*). В реальных устройствах напряжение низкого уровня — это 0 Вольт, а напряжение высокого уровня различно для конкретных устройств. Например, микросхемы популярнейшей серии К155 (и их зарубежного аналога — серии 7400), выпускавшиеся с 1960-х годов, имели напряжение высокого уровня +5 В; а современные процессоры фирмы Intel — около +1,3 В. Обратите внимание, что все потенциалы задаются относительно выбранного нулевого потенциала. На практике это, как правило, контакт, соединённый с одним из выводов источника питания (чаще — с «минусом»). В литературе этот контакт называют по-разному: «*сигнальная земля*», «земля», «общий провод»; в англоязычной — «Ground», «GND». При этом физическое заземление этого контакта вовсе не обязательно. Если вы внимательно посмотрите на печатную плату какого-нибудь цифрового устройства (например, материнской платы) со стороны дорожек, то наверняка увидите дорожку, «опоясывающую» всю плату. Это и есть сигнальная земля. Этот контакт почти всегда подключен к корпусу устройства.

В дальнейшем, когда мы будем создавать схемы цифровых устройств в Logisim, вы увидите, что сигнальная земля никогда не присутствует непосредственно в схеме: наличие этого контакта (одного и только одного) всегда подразумевается; схема не может функционировать без него. При физической реализации, все устройства, размещённые на схеме, подключаются к сигнальной земле (как и к источнику сигнала высокого уровня; чаще всего это «плюс» источника питания).

Вообще, важно запомнить очень простую идею: сигнал низкого уровня на контакте означает, что этот контакт соединён с «минусом» питания (то есть сопротивление между этим контактом и «минусом» питания очень мало), а сигнал высокого уровня означает, что контакт соединён с «плюсом» питания. Для некоторых ситуаций это может быть неверно, но в рамках данного курса мы с такими ситуациями не столкнёмся, поэтому помнить эту идею весьма полезно. Например, зная это, можно ответить на вопрос «что будет, если соединить контакты, на одном из которых 0, а на другом — 1?» Очевидно, что произойдёт *короткое замыкание*, и устройство перегорит.

В реальных устройствах могут присутствовать компоненты, использующие для работы разные напряжения питания (а значит, имеющие разные напряжения логической 1). Для совместной работы таких компонентов используются различные согласующие элементы. Поскольку Logisim — образовательный инструмент, в нём для простоты всегда рассматривается ситуация, когда напряжение высокого уровня одинаково для всех компонентов в схеме.

Логические 0 и 1 — конечно же абстракции. На практике напряжение на каком-либо контакте может отличаться от заданных значений низкого или высокого уровня. Небольшие отклонения допустимы (их диапазон задаётся производителями устройств). Существенные отклонения приводят к тому, что уже невозможно сказать, какой сигнал в данный момент на контакте — логический 0 или 1. В таком случае говорят, что значение *не определено*. Кроме того, существуют ситуации, когда сопротивление между контактом и остальной схемой очень велико. Такое состояние называют «*высокоимпедансным*», «Z-состоянием», или «плавающим контактом». Очевидно, что и в этом случае нельзя сказать, какое значение на контакте — 0 или 1. Такие ситуации, как правило, создают намеренно; об этом мы поговорим позднее. Logisim умеет работать с состоянием неопределённости, поэтому при проектировании схем мы будем уделять внимание таким состояниям.

## 1.2. Булевы функции и логические элементы

Булевы функции также называют *логическими функциями* или *функциями алгебры логики*, а применительно к цифровой электронике также *логическими операциями* или *битовыми операциями*. Существует строгое формальное определение булевой функции, с ним вы можете ознакомиться (или уже ознакомились) в курсе математической логики, однако здесь оно будет немного неуместным: в данном курсе нас

интересует сугубо практическое использование булевых функций. Поэтому здесь приводится более простое и наглядное объяснение.

Для начала оговоримся, что термин «*значение*» в данном разделе означает исключительно *логическое значение*, то есть, например, фраза «значение переменной» означает, что переменная может быть равна либо логическому 0, либо логической 1.

*Булева функция* от n переменных — это некоторое значение, которое полностью определяется (зависит от них) значениями переменных (переменные тоже являются логическими значениями). Для задания булевой функции обычно используют *таблицы истинности*. Каждая строка такой таблицы содержит одну из всех возможных комбинаций значений переменных и соответствующее значение функции. В такой таблице всегда $2^n$ строк. Количество всех n-арных булевых функций равно $2^{2^n}$, то есть существует 4 различных булевых функции от одной переменной (унарных) и 16 различных функций от двух переменных (бинарных).

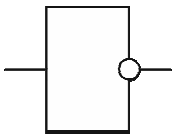
Семь из этих двадцати функций имеют особое значение в цифровой электронике: для физической реализации соответствующих им битовых операций существуют специальные устройства — *логические элементы* или *логические вентили*. Физически они реализованы на транзисторах в составе интегральных схем. Логические элементы являются минимальными составляющими при проектировании почти для любого цифрового устройства, своего рода «атомами» цифровой техники. Даже самый сложный процессор в конечном итоге состоит из логических элементов. Именно поэтому знание логических элементов — их названий, обозначений и поведения просто необходимо для прохождения данного курса.

Ниже приводится сводная таблица логических элементов (таблица 1.1). В ней даны различные варианты названий элементов, таблицы истинности, два варианта графических обозначений, а также мнемонические правила для запоминания поведения элементов.

*Таблица 1.1.* Логические элементы

**НЕ, NOT, логическое отрицание, инверсия, дополнение, негация**

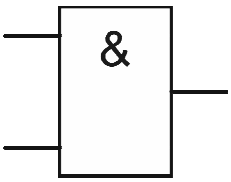
| x | f(x) |
|---|---|
| 0 | 1 |
| 1 | 0 |

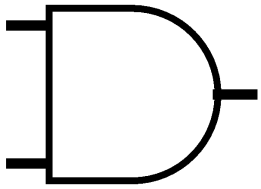


На выходе «1» тогда и только тогда, когда на входе «0»

**И, AND, логическое умножение, конъюнкция**

| x | y | f(x,y) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

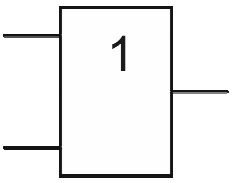


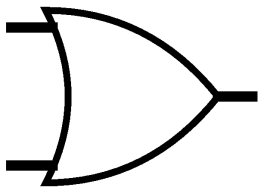



На выходе «1» тогда и только тогда, когда на всех входах «1»

**ИЛИ, OR, логическое сложение, дизъюнкция**

| x | y | f(x,y) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

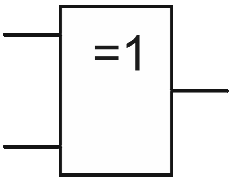


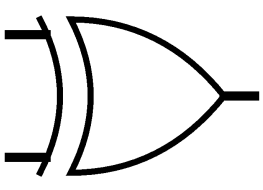



На выходе «1» тогда и только тогда, когда хотя бы на одном входе «1»

**Исключающее ИЛИ, XOR, сложение по модулю 2**

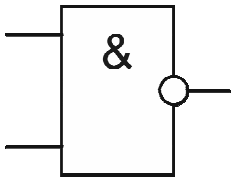
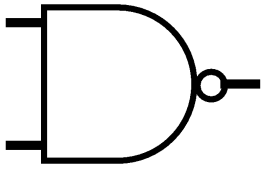
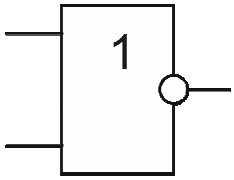
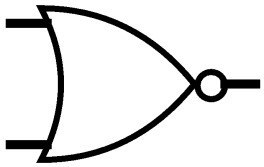
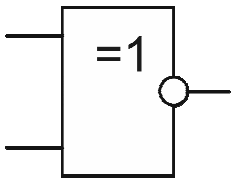
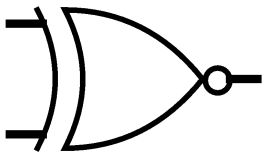
| x | y | f(x,y) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



На выходе «1» тогда и только тогда, когда ровно на одном входе «1»

*Окончание таблицы 1.1*

<table>
<tr><td colspan="3">И-НЕ, NAND, антиконъюнкция, штрих Шеффера</td></tr>
<tr><td>
<table>
<tr><td>x</td><td>y</td><td>f(x,y)</td></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td><td> 
</td><td>  </td></tr>
<tr><td colspan="3">На выходе «1» тогда и только тогда, когда хотя бы на одном входе «0»</td></tr>
<tr><td colspan="3">ИЛИ-НЕ, NOR, антидизъюнкция, стрелка Пирса</td></tr>
<tr><td>
<table>
<tr><td>x</td><td>y</td><td>f(x,y)</td></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td><td> 
</td><td>  </td></tr>
<tr><td colspan="3">На выходе «1» тогда и только тогда, когда на всех входах «0»</td></tr>
<tr><td colspan="3">Исключающее ИЛИ-НЕ, XNOR, эквивалентность</td></tr>
<tr><td>
<table>
<tr><td>x</td><td>y</td><td>f(x,y)</td></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td><td> 
</td><td>  </td></tr>
<tr><td colspan="3">На выходе «1» тогда и только тогда, когда «1» на любом количестве входов, отличном от одного</td></tr>
</table>

В таблице 1.1 приведены таблицы истинности для логических элементов с двумя входами (за исключением элемента НЕ), то есть для элементов, реализующих бинарные булевы функции. Однако эти элементы могут иметь большее количество входов; в таком случае они будут реализовывать булевы функции от большего количества переменных, а таблицы истинности будут определяться из тех же мнемонических правил, являющихся универсальными. Часто чтобы указать на конкретное количество входов элемента, название элемента записывают с предшествующим числом. Например, пятой позицией в таблице выше изображён элемент, полное название которого 2И-НЕ. Если бы он имел пять входов вместо двух, то полное название было бы 5И-НЕ.

Существует некоторая путаница с поведением элементов Исключающее ИЛИ и Исключающее ИЛИ-НЕ с количеством входов большим двух. Некоторые специалисты считают, что их правильное поведение должно описываться мнемоническим правилом «На выходе "1" тогда и только тогда, когда на нечётном количестве входов "1"» для элемента Исключающее ИЛИ и правилом «На выходе "1" тогда и только тогда, когда на чётном количестве входов "1"» для элемента Исключающее ИЛИ-НЕ. Поведение, указанное в таблице выше, основывается на стандарте IEEE 91; оно же является поведением по умолчанию в Logisim, хотя Logisim позволяет настраивать это поведение.

В приведённой выше таблице изображено по два варианта графических обозначений для каждого логического элемента. «Прямоугольные» обозначения более популярны в Европе, и являются стандартом в России, а «фигурные» обозначения более популярны в США. Logisim позволяет выбирать, какой из этих стандартов использовать. Если вы считаете, что вам стоит привыкнуть к российскому стандарту, то конечно, лучше использовать «прямоугольные» обозначения. В противном случае лучше использовать «фигурные» обозначения — потому что многие другие компоненты в Logisim имеют прямоугольную форму, и таким образом можно избежать путаницы и повысить удобочитаемость схемы.

Семью вышеуказанными простейшими логическими операциями (функциями), и даже некоторыми их подмножествами, можно выразить любые другие логические операции (то есть сколь угодно сложные булевы функции, зависящие от любого числа переменных). Такой набор простейших функций называется *функционально полным логическим базисом*. Таких базисов четыре:

- И, НЕ (2 элемента)
- ИЛИ, НЕ (2 элемента)
- И-НЕ (1 элемент)
- ИЛИ-НЕ (1 элемент)

На практике это имеет огромное значение: часто гораздо экономичнее реализовать устройство на однотипных элементах. Если же речь идёт о проектировании интегральных схем, то в подавляющем большинстве случаев используются только элементы И-НЕ, ИЛИ-НЕ и НЕ, так как при реализации на транзисторах остальные элементы фактически строятся из этих трёх.

## 1.3. Комбинационные и последовательностные логические устройства

Все цифровые устройства, не участвующие во вводе или выводе информации, можно разделить на две категории: комбинационные и последовательностные. Здесь под «устройством» понимается *логическое устройств*о, то есть не готовый к эксплуатации пользователем прибор, а составная единица такого прибора: например, процессор или модуль оперативной памяти; или ещё более мелкая составная часть (существующая как отдельный элемент только при проектировании, но не при реализации): регистр, обособленная группа логических элементов, и т.п.

*Комбинационное* логическое устройство — логическое устройство, выходные сигналы которого однозначно определяются входными сигналами (*комбинацией* сигналов на входах).

*Последовательностное* логическое устройство — логическое устройство, выходные сигналы которого определяются не только сигналами на входах, но и предысторией их работы, то есть состоянием элементов памяти (*последовательностью* сигналов на входах).

Таким образом, комбинационное устройство просто реализует определённую булеву функцию от значений на входах (или множество булевых функций, если устройство имеет несколько выходов); а последовательностное устройство имеет внутреннюю память, и фактически реализует булеву функцию от значений на входах и от значений, сохранённых в памяти. Нетрудно понять, что устройство, полученное соединением комбинационного и последовательностного устройств, всегда будет последовательностным.

Понимание этого разделения логических устройств очень важно для дальнейшего освоения курса. Давайте попробуем посмотреть с этой

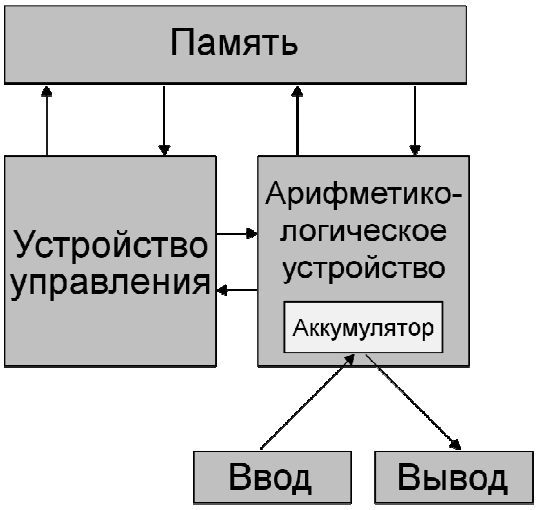



*Рис. 1.1.* Архитектура фон Неймана

позиции на схему архитектуры фон Неймана (рис. 1.1). Как уже было отмечено выше, устройства ввода и вывода не попадают ни в одну из категорий.

Память и аккумулятор — последовательностные устройства, так как способны сохранять информацию. Устройство управления и арифметико-логическое устройство (без аккумулятора) — комбинационные устройства, они только обрабатывают информацию, не храня её (а значит, они всего лишь реализуют набор булевых функций, и могут быть построены из логических элементов!). Однако, как только мы добавляем к арифметико-логическому устройству аккумулятор, оно становится последовательностным. Этот несложный анализ архитектуры компьютера расставляет всё по своим местам, и задача проектирования ЭВМ уже не выглядит такой невероятно сложной.

## 1.4. Минимизация булевых функций

При проектировании цифровых устройств нужно помнить, что каждый новый логический элемент увеличивает общий размер кристалла интегральной схемы, потребляет электроэнергию, увеличивает время распространения сигнала, а значит, увеличивает стоимость и ухудшает свойства устройства. Таким образом, задача инженера — не только спроектировать работающее устройство, но и снизить количество присутствующих в нём логических элементов насколько это возможно.

Как было сказано выше, любое комбинационное логическое устройство реализует набор булевых функций (их будет столько, сколько выходов у устройства) от значений сигналов на его входах. Чтобы снизить количество логических элементов, из которых состоит комбинационное устройство, нужно произвести *минимизацию булевых функций*, которые оно реализует.

Рассмотрим процесс минимизации на простом примере. Допустим, нам нужно спроектировать устройство, имеющее три входа (a, b, c) и один выход (out). На выходе должна быть логическая 1 тогда, и только тогда, когда 1 поступает на количество входов, меньшее двух. В остальных случаях на выходе должен быть 0. Составим таблицу

истинности для соответствующей булевой функции (таблица 1.2).

*Таблица 1.2.* Таблица истинности для примера

| a | b | c | out |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

До этого момента мы рассматривали только один способ задания булевой функции — таблицу истинности. Однако очевидно, что составить схему реализации функции на логических элементах непосредственно из таблицы истинности нельзя. Для этой цели, как правило, используются логические выражения, которые можно составить несколькими способами, зная таблицу истинности. *Логическое выражение* (*булева формула*) — выражение, в состав которого входят логические переменные, связанные знаками логических операций (*пропозициональных связок*), результат вычисления которого — также логическое значение. Из логических операций чаще всего используются НЕ («НЕ x» может обозначаться как «$\overline{x}$», «¬x», «x'» или «~x»), И («x И y» может обозначаться как «x&y», «x·y», «xy» или «$x \wedge y$»), и ИЛИ («x ИЛИ y» может обозначаться как «$x \vee y$», «x | y», или «x+y»), хотя другие операции иногда тоже используются.

Один из самых простых и часто используемых способов составить логическое выражение из таблицы истинности — использовать совершенную дизъюнктивную нормальную форму (СДНФ). *Дизъюнктивная нормальная форма* (ДНФ) — способ записи логического выражения, при котором булева формула имеет вид дизъюнкции нескольких конъюнкций (если вспомнить, что дизъюнкция (ИЛИ) — это логическое сложение, а конъюнкция (И) — логическое умножение, то можно сказать, что ДНФ — это сумма произведений). *Совершенная дизъюнктивная нормальная форма (СДНФ)* — это ДНФ, удовлетворяющая трём условиям: в ней нет одинаковых элементарных конъюнкций; в каждой конъюнкции нет одинаковых пропозициональных букв (переменных); каждая элементарная конъюнкция содержит каждую пропозициональную букву из входящих в данную ДНФ пропозициональных букв, причём в одинаковом порядке.

Для конъюнкции (логического произведения) переменных, входящих в логическое выражение, или их отрицаний существует термин *минтерм*. Легко заметить, что минтерм принимает значение «1» при единственном из всех возможных наборов значений аргументов.

Составление СДНФ для заданной таблицы истинности

осуществляется следующим образом. Каждой строке таблицы, которая имеет «1» в последнем столбце (то есть на выходе функции), соответствует один минтерм СДНФ. При этом если в данной строке переменная имеет значение «1», то она непосредственно присутствует в минтерме, а если «0» — то в минтерм входит её отрицание. Таким образом, СДНФ, соответствующая таблице истинности проектируемого нами устройства, будет выглядеть следующим образом:

$$\bar{\mathbf{a}}\,\bar{\mathbf{b}}\,\bar{\mathbf{c}} + \bar{\mathbf{a}}\,\bar{\mathbf{b}}\,\mathbf{c} + \bar{\mathbf{a}}\,\mathbf{b}\,\bar{\mathbf{c}} + \mathbf{a}\,\bar{\mathbf{b}}\,\bar{\mathbf{c}}$$

Для реализации этого выражения на логических элементах нам потребуется 9 элементов НЕ, по одному элементу И для каждого минтерма и один элемент ИЛИ для дизъюнкции (рис. 1.2). Отметим, что обозначение входов квадратами, а выходов кружками специфично для Logisim и не является стандартом. Как видите, здесь используются логические элементы с количеством входов большим двух. Одна из задач минимизации — по возможности избегать присутствия таких габаритных элементов.

Теперь приступим непосредственно к минимизации. *Минимизация булевой функции* — получение логического выражения для булевой функции (или набора функций), содержащего минимальное количество логических операций. Существует несколько способов минимизации: применение тождеств булевой алгебры к СДНФ; применение определённых алгоритмов к таблице истинности: карты Карно, метод Куайна, метод Куайна — Мак-Класки, алгоритм Espresso, и т.д. Подробные описания этих способов весьма длинны, и не приводится в данном пособии; вы с лёгкостью найдёте их самостоятельно. Уметь непосредственно применять эти методы довольно полезно, однако на практике (когда функции поистине огромны) применяют специализированные программы, реализующие эти методы. В рамках лабораторных работ данного курса нам будет вполне достаточно встроенной в Logisim реализации метода Куайна — Мак-Класки, а для более объёмных функций, с которыми вы можете столкнуться при выполнении курсовой работы, одним из решений может быть использование бесплатной программы для MS Windows, реализующей алгоритм Espresso — Logic Friday (http://sontrak.com/). Для декомпозиции

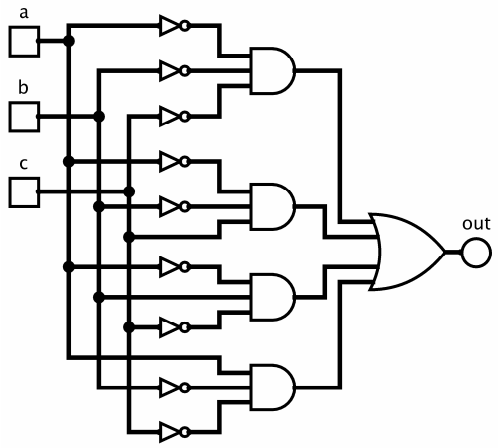



*Рис. 1.2.* Реализация непосредственно СДНФ

минимизированных функций и построения готовой схемы Logic Friday использует другую программу — misII.

В случае простых функций (вроде той, которую мы реализуем в нашем текущем примере) все эти алгоритмы, как правило, будут давать один и тот же результат (логическое выражение). Однако, при минимизации больших функций, или набора функций, зависящих от одних и тех же переменных, разные алгоритмы могут давать различные результирующие логические выражения. Кроме того, при решении практических задач может быть необходимо построить реализацию функции на однотипных элементах (используя какой-либо базис, например, И-НЕ). Программы, реализующие минимизацию булевых функций, как правило, имеют возможность задавать такие ограничения.

Для нашего примера любой из вышеперечисленных алгоритмов выдаст такое минимизированное логическое выражение:

$$\overline{\mathbf{a}}\,\overline{\mathbf{b}} + \overline{\mathbf{a}}\,\overline{\mathbf{c}} + \overline{\mathbf{b}}\,\overline{\mathbf{c}}$$

Реализация этого выражения на логических элементах с дополнительно указанным условием использовать только двухвходовые элементы выглядит следующим образом (рис. 1.4). Количество и размеры элементов по сравнению с предыдущей реализацией заметно сократились.

Logic Friday выдаст то же выражение, однако предложит ещё более простую реализацию этой булевой функции на логических элементах (рис. 1.3). Этой программе можно задать другой набор доступных для использования логических элементов, и результат окажется иным. Какой из вариантов реализации выбрать, определяется конкретной практической задачей.

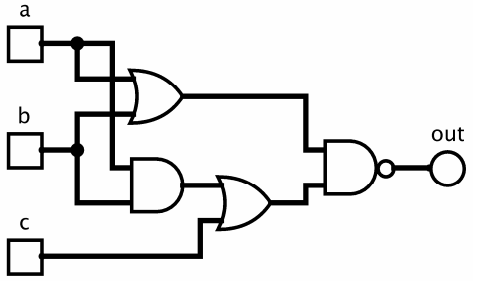



*Рис. 1.3.* Реализация, предложенная Logic Friday

Существует ещё один важный момент при минимизации функций, реализуемых комбинационными логическими устройствами. Часто возникают ситуации, когда нам известны комбинации входных сигналов, которые *никогда* не поступят на входы устройства. Это может случиться, например, когда входные сигналы — это биты многобитного значения, являющегося неким

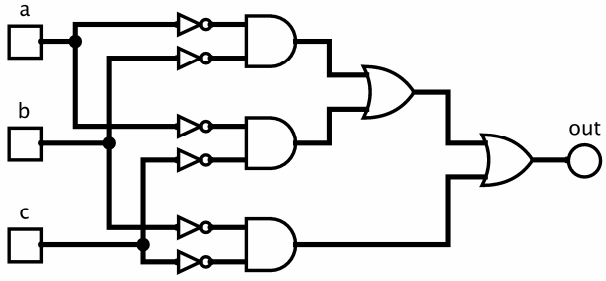



*Рис. 1.4.* Реализация минимальной ДНФ

кодом, и не все конкретные многобитные значения этого кода имеют ассоциированные с ними смысловые значения. Допустим, что для нашего примера известно, что на входы a и b никогда не поступят одновременно логические 1. Это означает, что нам *не важно*, что будет на выходе устройства в такой ситуации — она просто не может случиться. Чтобы отразить это в таблице истинности, используется символ «x» (в англоязычной литературе он имеет смысл «*don't care*», соответствующий нашему «не важно»). Таблица истинности для нашего устройства изменится следующим образом (таблица 1.3).

*Таблица 1.3.*
Таблица истинности, содержащая «don't care»

| a | b | c | out |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | x |
| 1 | 1 | 1 | x |

Метод Куайна — Мак-Класки и алгоритм Espresso позволяют использовать «x» в таблицах истинности, и наличие таких значений *весьма существенно* уменьшает количество необходимых для реализации логических элементов.

Ниже приводятся логическое выражение для новой таблицы истинности, а также непосредственная реализация из этого выражения (рис. 1.5) и реализация, предложенная Logic Friday (рис. 1.6).

$$\bar{\mathbf{a}}\,\bar{\mathbf{b}} + \bar{\mathbf{c}}$$

Заметьте, что наличие «x» в некоторых строках таблицы истинности не означает, что на выходе устройства будет что-то неопределённое. Напротив, при поступлении на входы комбинации сигналов, которую мы не рассматривали как возможную, на выходе будет вполне конкретное значение. Это будет такое значение, которое позволило алгоритму минимизации найти минимальное логическое выражение. Иными словами, добавляя значения «don't care» в таблицу истинности, мы как бы придаём дополнительную гибкость нашей функции, позволяющую алгоритму находить более короткие минимальные выражения.

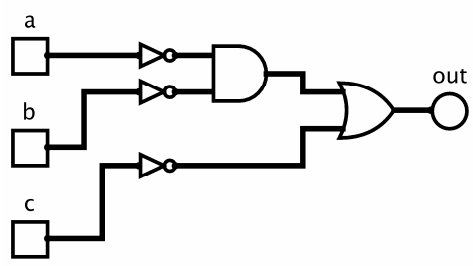



*Рис. 1.5.* Реализация непосредственно ДНФ

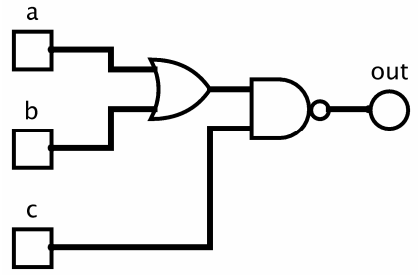



*Рис. 1.6.* Реализация, предложенная Logic Friday

Здесь мы рассмотрели минимизацию одной функции. На практике гораздо чаще возникает необходимость минимизировать набор булевых функций от одних и тех же входных сигналов. Алгоритмы для минимизации в таких случаях будут принимать на входе таблицу истинности, в которой каждой входной переменной и каждой функции соответствует по столбцу, а на выходе давать несколько логических выражений (по количеству функций), или готовую схему, включающую логические элементы, и имеющую количество выходов, равное количеству функций.

## 1.5. Задержки устройств

До этого момента мы никак не затрагивали вопросы, связанные с изменением сигнала во времени. Тем не менее, это существенная часть всего процесса проектирования цифровых устройств, в особенности, при физической реализации. В данном курсе мы, конечно, не дойдём до физической реализации устройств, а будем заниматься проектированием на «логическом» уровне. Однако уделить внимание этому вопросу всё же очень важно.

После того, как на входе устройства (например, логического элемента) уровень сигнала изменился, значение на выходе обновится только через некоторый промежуток времени. Этот промежуток времени называется *задержкой*. Задержка обуславливается тем, что транзисторам в составе логических элементов нужно время, чтобы открыться или закрыться. Для справки, время задержки логических элементов для уже упоминавшейся серии микросхем К155 — около 20нс, а для современных процессоров компании Intel время задержки измеряется сотыми долями наносекунды.

Посмотрим ещё раз на схему, изображённую на рисунке 1.2. При распространении сигнала от входа к выходу этой схемы, он может последовательно «пройти» через 2 или 3 логических элемента в зависимости от «маршрута». Это значит, что в худшем (самом медленном) случае общая задержка распространения сигнала будет равна сумме задержек каждого из этих трёх элементов. Таким образом, последовательное соединение большого количества логических элементов делает устройство в целом более медленным, поэтому ещё один (и весьма важный!) критерий при выборе варианта минимизации булевой функции — количество элементов в самой длинной цепи последовательно соединённых элементов, идущей от входа к выходу. Сравнивая, например, реализации, представленные на рисунках 1.4 и 1.3,

предпочтение нужно отдать схеме на рисунке 1.3, так как самая длинная цепь в этой схеме состоит из трёх элементов, а на рисунке 1.4 — из четырёх.

Обратите внимание, что равные задержки у разных логических элементов — это упрощение. На практике логические элементы реализуются на транзисторах, и их фактическая задержка зависит от числа последовательно соединённых транзисторов в составе элемента, и ещё от некоторых факторов.

## 1.6. Тактовые импульсы. Синхронные и асинхронные устройства

Как правило, мы подразумеваем, что цифровое устройство должно выполнять какие-то операции с течением времени. Однако очевидно, что рассмотренные нами комбинационные логические устройства не способны делать это сами по себе: они только реагируют на изменение состояния на входах. Но что должно менять во времени эти состояния? Очевидно, что должно существовать устройство, задающее «ход времени» для схемы. В цифровой технике такое устройство называется *тактовым генератором*. В первом приближении можно рассматривать его как устройство с единственным выходом, сигнал на котором непрерывно меняется с 0 на 1 и обратно. Иными словами, этот сигнал имеет прямоугольную форму, а сменяющие друг друга 0 и 1 называют *тактовыми* или *синхронизирующими импульсами*. Отрезок времени, в течение которого сигнал на выходе тактового генератора остаётся постоянным, называют *тактом*. Количество импульсов в единицу времени — *тактовой частотой*. Внутреннее устройство тактового генератора в данном курсе мы рассматривать не будем — оно уже рассматривалось в курсе электроники; в простейшем случае это мультивибратор.

Изменение сигнала на выходе тактового генератора с 0 на 1 тоже не происходит мгновенно. Однако в первом приближении мы будем считать, что это время переключения много меньше продолжительности одного такта, поэтому момент переключения сигнала можно считать точечным моментом времени. Момент изменения значения с 0 на 1 называется *передним фронтом*, а с 1 на 0 — *задним фронтом*.

Существует ещё одна классификация цифровых устройств — на синхронные и асинхронные. *Асинхронные устройства* изменяют состояние на выходе (и внутреннее состояние, если это устройство

памяти) сразу после изменения сигнала на входе (с поправкой на задержку, разумеется). *Синхронные устройства* изменяют состояние на выходе (и внутреннее состояние) в момент, когда на специальный вход, называемый *тактовым* (или *синхронизирующим*), поступает очередной тактовый импульс, причём такое устройство может реагировать как на передний фронт, так и на задний. Синхронными, как правило, делают только последовательностные устройства (элементы памяти), хотя теоретически ничто не препятствует делать синхронными комбинационные устройства.

Причина потребности в синхронных устройствах — наличие задержек у компонентов. Очевидно, что если законченное устройство последовательно выполняет какие-то операции, то оно наверняка использует результаты предыдущих операций как исходные данные для последующих, а значит его выходы каким-то образом связаны с его входами. Иными словами, схема такого устройства как бы замкнута в кольцо. Если подавать тактовые импульсы в какую-то точку этого кольца, то из-за задержек в произвольный момент времени будет сложно сказать, среагировал ли конкретный элемент схемы на новый импульс, или его состояние ещё не обновилось. Можно представить себе это как волну обновлений состояний, проходящую вдоль этого кольца. Но поскольку разные ветви кольца могут иметь разное количество элементов (а значит и разные значения суммарной задержки), в какой-то момент времени на входы определённого элемента могут прийти два сигнала, один из которых уже обновлён (волна «дошла» до этого входа), а второй — нет. Тогда на выходе элемента возникнет ложное значение, и это пустит по кольцу новую волну. Про такие ситуации говорят, что устройство *возбуждается*. Этого можно избежать, если в какой-то точке кольца поместить синхронное устройство. Такое устройство будет дожидаться, пока волна обновлений пройдёт через все элементы схемы, а затем тактовый импульс даст сигнал для обновления выходов этого устройства, которое запустит новую волну, и т.д. Возникает закономерное предположение, что продолжительность одного такта должна быть больше, чем суммарная задержка самой длинной ветви цепи, соединяющей выход синхронного устройства и его вход. Это обязательное условие для любого устройства, в котором имеется генератор тактовых импульсов. Logisim моделирует задержки компонентов (правда, не совсем реалистично для некоторых случаев), и при помещении на схему тактового генератора, Logisim гарантирует, что состояние выходов всех компонентов схемы обновится до генерации очередного тактового импульса.

Для наглядности рассмотрим вопросы, связанные с тактовыми импульсами, на простом практическом примере. Допустим, что нам нужно спроектировать устройство с четырьмя выходами, к которым будут подключены светодиоды, и устройство будет заставлять эти четыре светодиода загораться поочерёдно в цикле. Иными словами, логическая 1 должна появляться на выходах (назовём их a, b, c, d) в таком порядке: a, b, c, d, a, b, c... и т.д. В каждый конкретный момент времени устройство будет находиться в одном из четырёх состояний, а значит, нам нужен будет элемент памяти, способный «помнить», в каком состоянии находится устройство в данный момент. Очевидно, что такой элемент должен хранить два бита данных. Мы будем использовать два примитивных элемента памяти — два D-триггера, каждый из которых хранит по одному биту. Нас интересуют три контакта D-триггера: выход Q, на который всегда поступает значение, сохранённое в данный момент в триггере; вход D, на который подаётся новое значение для сохранения; и тактовый вход (он обозначается треугольником). Значение со входа D записывается в триггер в момент, когда на тактовый вход поступает тактовый импульс (передний фронт). Таким образом, D-триггер — синхронное устройство.

Кроме тактового генератора и двух D-триггеров, нам понадобится комбинационное управляющее устройство, которое мы построим на логических элементах. У этого устройства должно быть два входа (bit0 и bit1), на которые всегда будут поступать два значения, сохранённые в триггерах, 4 выхода на светодиоды и 2 выхода (new0 и new1), которые будут подавать новые значения для записи в триггеры (с поступлением каждого нового тактового импульса код состояния устройства, хранимый в триггерах, должен изменяться на новый). Состояния a, b, c, d имеют двухбитные коды соответственно 00, 01, 10, 11.

Составим таблицу истинности для этого устройства управления (таблица 1.4). После минимизации этих шести булевых функций и реализации их на логических элементах с помощью встроенной в Logisim реализации метода Куайна — Мак-Класки, получим представленную ниже схему устройства управления (рис. 1.7).

*Таблица 1.4.* Таблица истинности устройства управления

| bit1 | bit0 | a | b | c | d | new1 | new0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

Далее подключим D-триггеры (они обозначены прямоугольниками с буквами D и Q) к соответствующим входам и выходам

устройства управления, а тактовый генератор (он обозначен условным изображением прямоугольного сигнала). Полная схема устройства готова (рис. 1.8). Если составить эту схему в Logisim и запустить моделирование, то мы увидим желаемый результат — светодиоды будут циклически переключаться.

Рассмотрим порядок работы этой схемы. В исходном состоянии оба триггера хранят нули, и в соответствии с таблицей истинности горит светодиод a. Как только тактовый генератор выдаст новый импульс, триггеры сохранят новые значения, поступающие из схемы на их входы D. Это изменит значения на их выходах, и по цепи пройдёт «волна» обновлений — логические элементы поочерёдно обновят сигналы на выходах. Когда это волна дойдёт до триггеров, на их входы D поступят новые значения, но триггеры не сохранят их до поступления нового тактового импульса. Далее процесс будет продолжать в том же порядке, пока от тактового генератора поступают импульсы.

Обратите внимание на два провода, идущих ко входам схемы от выходов триггеров. Это как раз те провода, которые превращают схему в замкнутое кольцо. Если бы триггеры не были синхронными устройствами, то схема начала бы возбуждаться.

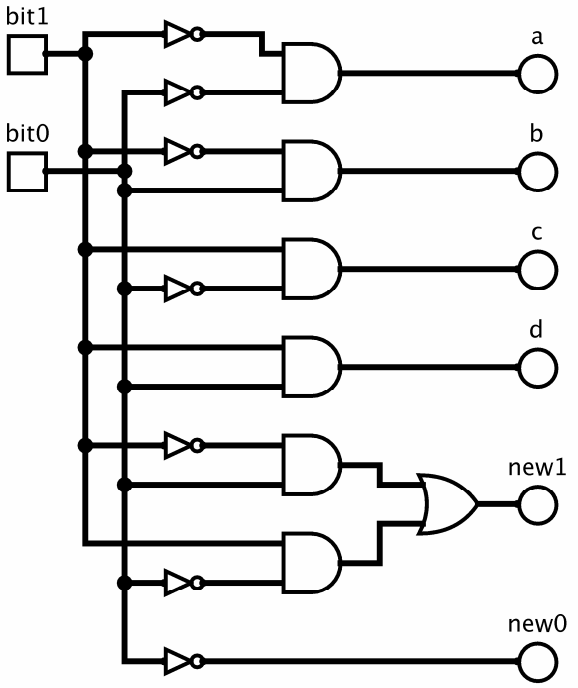



*Рис. 1.7.* Устройство управления

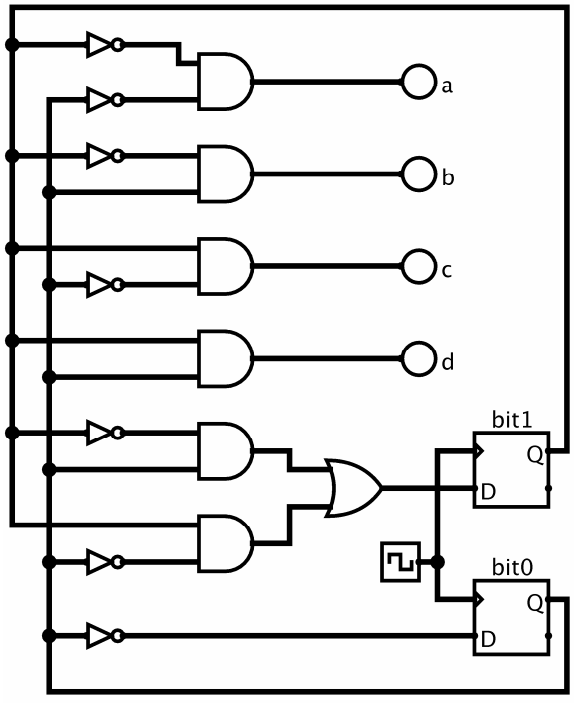



*Рис. 1.8.* Полная схема устройства

# 2. Программа Logisim

## 2.1. Знакомство с программой

Logisim — это образовательный инструмент для разработки и моделирования цифровых логических схем. Её написал и продолжает активную разработку преподаватель Hendrix College, США, Carl Burch. Программа написана на Java и требует для работы установленную Java Runtime Environment. На момент выхода данного пособия последняя стабильная версия Logisim — 2.7.1, однако рекомендуется использовать тестовую версию 2.7.2.251, так как в ней исправлены некоторые существенные ошибки. Если же доступна более новая стабильная версия, то стоит использовать её. Официальная страница программы на русском языке: http://cburch.com/logisim/ru

Logisim является свободным программным обеспечением и распространяется на условиях лицензии GNU GPL. Это означает, что вы можете свободно использовать и распространять её в любых целях, читать и редактировать её исходный код, и распространять модифицированные версии программы на тех же условиях.

Интерфейс Logisim полностью переведён на русский язык, программа распространяется с подробнейшей справочной системой на русском языке. Данная глава пособия содержит только самые основные сведения о программе, необходимые для быстрого знакомства с ней. Почти все нюансы работы с программой описаны в справочной системе — от получения базовых навыков работы до подробного описания всех компонентов встроенной библиотеки.

Интерфейс программы состоит из строки меню, панели инструментов, панели проводника, таблицы атрибутов и холста. *Панель проводника* содержит список схем проекта и все инструменты загруженных библиотек. Любые инструменты можно выносить на *панель инструментов*. Состав панели инструментов настраивается в окне параметров проекта, которое вызывается из пункта «Параметры» меню «Проект». На *холсте* располагаются все компоненты редактируемой в данный момент схемы. Холст покрыт сеткой, по которой выравниваются все компоненты и провода схемы. *Таблица атрибутов* содержит атрибуты выделенного в данный момент инструмента или компонента на холсте.

Лучший способ быстро освоить базовые навыки работы с Logisim —

пройти все шаги «Пособия начинающего» из меню «Справка».

Logisim позволяет не только редактировать цифровые схемы, но и моделировать их поведение. Если флаг «Моделирование включено» из меню «Моделировать» установлен, то Logisim просчитывает процессы, происходящие в схеме прямо во время её редактирования: меняются значения на входах и выходах устройств, обновляется внутреннее состояние устройств памяти, устройства вывода отображают соответствующую информацию, а провода меняют цвет в зависимости от проходящих по ним значений. Для схем, на которых присутствуют тактовые генераторы, можно либо осуществлять моделирование потактово (пункт «Один такт» из меню «Моделировать»), либо установить максимальную тактовую частоту (пункт «Тактовая частота») и запустить продолжительное моделирование схемы (пункт «Такты включены»). Вернуть схему в исходное состояние можно с помощью пункта «Сбросить моделирование».

## 2.2. Подсхемы

Одна из важнейших возможностей Logisim — создание подсхем. Идея подсхем аналогична идее подпрограмм в программировании: они служат для разбиения большой работы на более мелкие части, для повторного использования уже спроектированных частей схем и для упрощения процесса отладки.

Добавить схему можно с помощью пункта «Добавить схему...» из меню «Проект». Одна из схем в проекте обязательно является главной — она открывается первой при загрузке проекта. После запуска моделирования будет выполняться моделирование схемы, просматриваемой в данный момент. Чтобы включить схему в качестве подсхемы в другую схему, нужно выделить её и расположить на холсте, как и любой другой элемент. При этом образуется иерархия вложенных подсхем, верхним уровнем которой является просматриваемая схема. Чтобы редактировать схему, нужно дважды щёлкнуть на её названии в панели обозревателя. Располагать несколько копий одной схемы — совершенно нормальная практика. При этом редактирование этой схемы будет приводить к изменениям во всех копиях, но моделирование поведения каждой копии происходит автономно.

Передача сигналов между какой-либо схемой и расположенной в ней подсхемой происходит через контакты. Контакты бывают входные и выходные. По умолчанию на панели инструментов присутствуют оба вида контактов. Также компонент «Контакт» есть во встроенной

библиотеке «Проводка». В цифровой технике при соединении элементов проводами всегда определено, какой элемент передаёт сигнал, а какой принимает. Входной контакт подсхемы будет принимать сигнал из внешней схемы и передавать его в подсхему. Выходной контакт подсхемы будет передавать сигнал из подсхемы во внешнюю схему.

Контакты подсхемы при расположении её копии на холсте другой схемы будут располагаться с той стороны «корпуса» подсхемы, в какую сторону направлен контакт (атрибут «Направление») внутри подсхемы. Очень важно давать чёткие осмысленные имена контактам (атрибут «Метка»), тогда подключать к расположенной на холсте копии подсхемы провода будет очень легко.

Для верхней схемы в иерархии всегда показывается реальное текущее состояние во время моделирования. Однако чтобы наблюдать текущее состояние конкретной копии подсхемы, расположенной в иерархии ниже, нужно щёлкнуть правой кнопкой мыши на этой копии на холсте и выбрать в контекстном меню пункт «Рассмотреть...».

Logisim имеет небольшой встроенный редактор векторной графики для изменения внешнего вида подсхем при добавлении их в другие схемы. Он также позволяет изменять расположение контактов подсхемы. Этот редактор вызывается через пункт «Редактировать внешний вид схемы» из меню «Проект» или нажатием соответствующей кнопки на панели инструментов.

## 2.3. Провода

Располагать на холсте провода можно инструментами «Правка» или «Проводка». Провод и контакт элемента считаются соединёнными, если они располагаются строго в одной и той же точке сетки. То же относится и к соединению провода с другим проводом. При перемещении по холсту уже подключенных элементов, Logisim будет сохранять соединения, если это возможно. Этого можно избежать, если при перетаскивании нажать клавишу Shift.

Ещё одна важнейшая функция Logisim — возможность собирать провода в пучки. Пучки обозначаются на схемах чёрным цветом. Очень многие компоненты библиотеки имеют атрибут «Разрядность данных». Этот атрибут может принимать значение от 1 до 32, и если он больше 1, то подключение провода к соответствующему контакту компонента автоматически создаст пучок проводов. Естественно, нужно соединять контакты с одинаковой разрядностью данных. Разрядность данных можно назначать также входным и выходным контактам схем, что

значительно уменьшает количество проводов на схеме, а значит, повышает её удобочитаемость.

Когда возникает потребность разделить пучок проводов на отдельные провода или более мелкие пучки (или наоборот — собрать их в один пучок), нужно применить компонент «Разветвитель» из встроенной библиотеки «Проводка». Этот компонент также позволяет назначать в каком порядке провода входят в пучок — это настраивается в таблице атрибутов разветвителя.

Провода в Logisim могут иметь один из семи цветов, каждый из которых несёт определённую информацию о проводе.

- Серый — разрядность провода неизвестна (он не подключен ни к каким компонентам)
- Синий — провод несёт однобитный сигнал, но его состояние не определено.
- Тёмно-зелёный — провод несёт однобитный сигнал, в данный момент этот сигнал — «0».
- Светло-зелёный — провод несёт однобитный сигнал, в данный момент этот сигнал — «1».
- Чёрный — провод несёт многобитный сигнал (его разрядность больше единицы). При этом любое количество отдельных однобитных проводов в пучке могут иметь неопределённое состояние.
- Красный — провод содержит значение «ошибка». Это происходит, когда провод замыкает сразу несколько выходов компонентов, и на выходах разное значение сигнала. На практике это означает короткое замыкание.
- Оранжевый — провод подключён к выходам компонентов, имеющим разную разрядность.

Выделение цветом проводов, несущих «0» или «1» позволяет наблюдать изменение значений на выходах компонентов и эффективно отлаживать схему.

## 2.4. Комбинационный анализ

Ещё одна замечательная функция Logisim — возможность проводить анализ и синтез *комбинационных* логических устройств.

Если схема не содержит последовательностных логических устройств, контактов разрядностью больше 1 и количество входных и выходных контактов не больше 12, то для схемы можно провести комбинационный анализ. Это можно сделать через пункт «Анализировать схему» из меню

«Проект». В открывшемся окне «Комбинационный анализ» присутствуют вкладки «Входы», «Выходы», «Таблица», «Выражение» и «Минимизация». Вкладка «Таблица» содержит полную таблицу истинности, полученную для схемы; вкладка «Выражение» содержит список выражений каждой из булевых функций для выходных контактов. Вкладка «Минимизация» содержит минимизированные таблицы истинности и выражения для булевых функций. И выражения и таблицы истинности можно свободно редактировать, копировать и вставлять.

Открыв то же окно «Комбинационный анализ» из меню «Окно», можно ввести во вкладках «Входы» и «Выходы» имена входных и выходных контактов для будущей схемы; ввести таблицу истинности или выражения булевых функций, нажать кнопку «Построить схему», и указать название новой схемы. Logisim синтезирует схему, используя логические элементы.

## 2.5. Библиотека компонентов

Огромным преимуществом программы Logisim является обширная библиотека компонентов. Далее даётся краткое описание каждого компонента библиотеки. Если компонент является реализацией типичного для цифровой техники устройства, то никакого описания не приводится. Для получения подробной информации по каждому компоненту следует обратиться ко встроенной справке (пункт «Справка по библиотеке...» из меню «Справка»).

**Библиотека «Проводка»**

- Разветвитель — позволяет разделять пучки на отдельные провода и объединять отдельные провода в пучки.
- Контакт — передаёт значения между подсхемой и содержащей её внешней схемой.
- Датчик — отображает значение в точке схемы, к которой подключен. Позволяет выбирать основание системы счисления для отображения.
- Тоннель — позволяет соединять удалённые точки схемы без протягивания провода.
- Согласующий резистор — меняет в определённую сторону значение на проводе, к которому подключен, если значение на нём не определено.
- Тактовый генератор — формирует прямоугольные импульсы.
- Константа — источник постоянного значения

- Питание — источник логической единицы
- Земля — источник логического нуля
- Транзистор — полевой транзистор, работающий в режиме электронного ключа
- Передаточный вентиль — комбинация двух комплементарных полевых транзисторов
- Расширитель битов — преобразует значение в значение с другой разрядностью. При увеличении разрядности позволяет выбирать, чем заполнять старшие биты.

**Библиотека «Элементы»**

- Элемент НЕ
- Буфер — повторитель сигнала («элемент «ДА»)
- Элемент И
- Элемент ИЛИ
- Элемент И-НЕ
- Элемент ИЛИ-НЕ
- Элемент Исключающее ИЛИ
- Элемент Исключающее ИЛИ-НЕ
- Нечётность — на его выходе единица, если единица на начётном количестве входов
- Чётность — на его выходе единица, если единица на чётном количестве входов
- Управляемый буфер — позволяет создавать на его выходе неопределённое значение
- Управляемый инвертор — то же, что и управляемый буфер, но выполняющий инвертирование выхода.

**Библиотека «Плексоры»**

- Мультиплексор
- Демультиплексор
- Декодер
- Шифратор приоритетов — выдаёт номер старшего входа, на котором «1»
- Селектор битов — разделяет многоразрядный вход на несколько групп битов, и пускает на выход выбранную управляющим входом группу

**Библиотека «Арифметика»**

- Сумматор
- Вычитатель
- Множитель
- Делитель
- Отрицатель — выдаёт значение в дополнительном коде, противоположное по знаку входному значению
- Компаратор — сравнивает два числа
- Сдвигатель — реализация левого и правого сдвига
- Сумматор битов — выдаёт количество единиц, поступающих на его многоразрядные входы
- Искатель битов — выдаёт номер старшего или младшего нуля или единицы в многоразрядном входном значении.

**Библиотека «Память»**

- D-триггер
- T-триггер
- JK-триггер
- RS-триггер
- Регистр
- Счётчик
- Сдвиговый регистр
- Генератор случайных чисел
- ОЗУ
- ПЗУ

**Библиотека «Ввод/вывод»**

- Кнопка
- Джойстик
- Клавиатура — выдаёт ASCII коды введённых пользователем символов
- Светодиод
- 7-сегментный индикатор
- Шестнадцатеричный индикатор — 7-сегментный индикатор, выводящий шестнадцатеричное значение, поступающее на его 4-разрядный вход.
- Светодиодная матрица
- Терминал — отображает поступающие на его вход ASCII коды в

виде строк текста

**Библиотека «Базовые»**

- Инструмент Нажатие — инструмент для взаимодействия с компонентами схемы: нажимания кнопок, движения джойстика, установки значений контактов, просмотра состояния пучков проводов, и т.д.
- Инструмент Правка — основной инструмент для редактирования схем: перемещения компонентов и добавления проводов. Этот инструмент совмещает в себе функции четырёх инструментов, описанных ниже (можно считать их устаревшими).
- Инструмент Выбор — инструмент для выделения элементов схемы.
- Инструмент Проводка — инструмент для добавления проводов.
- Инструмент Текст — инструмент для добавления текстовых полей и редактирования меток компонентов.
- Инструмент Меню — инструмент для вызова контекстного меню компонентов.
- Метка — инструмент для добавления текстовых меток в любую точку холста.

## 2.6. Различные возможности

В этом параграфе приводится список различных дополнительных возможностей Logisim с краткими описаниями. Более подробную информацию о каждой возможности можно получить в соответствующем разделе справки.

Любой файл проекта Logisim можно загрузить в другой проект в качестве библиотеки. В таком случае все его схемы будут доступны как компоненты, но изменять их внутреннюю структуру будет нельзя. Это осуществляется через пункт «Загрузить библиотеку: Библиотека Logisim...» из меню «Проект».

Существует возможность писать на языке Java собственные библиотеки компонентов, обладающие всем набором функций, доступных в Logisim. Руководство по написанию таких библиотек — в разделе «Библиотеки JAR» руководства пользователя Logisim.

Моделировать поведение схем можно без запуска графического

интерфейса пользователя Logisim. Существует множество параметров для запуска и автоматизированного моделирования схем из командной строки. Подробнее — в разделе «Проверка из командной строки» руководства пользователя Logisim.

Растровые изображения схем можно экспортировать в файлы — как индивидуально, так и группами. Это осуществляется через пункт «Экспортировать изображение...» из меню «Файл».

Любой инструмент можно привязать к определённой комбинации клавиш Shift, Control и Alt и кнопок мыши. Это настраивается на вкладке «Мышь» окна параметров проекта (пункт «Параметры...» из меню «Проект»).

Любой файл проекта Logisim можно использовать как шаблон для создания новых проектов. Это удобно, если у вас есть файл проекта с настроенной панелью инструментов, поведением мыши, и т.д. Это осуществляется на вкладке «Шаблон» окна настроек приложения (пункт «Настройки...» из меню «Файл»).

Графические обозначения логических элементов можно выбрать на вкладке «Международные» окна настроек приложения.

Отображение фактической тактовой частоты моделирования можно включить на вкладке «Окно» окна настроек приложения.

Полную статистику по типам и количеству компонентов, содержащихся в схеме и входящих в её состав подсхемах можно получить через пункт «Получить статистику схемы» из меню «Проект».

Существует возможность просматривать в виде дерева иерархию подсхем и перемещаться по ней. Для перехода в такой режим просмотра нужно выбрать пункт «Показать дерево моделирования» из меню «Проект», или нажать соответствующую кнопку на панели инструментов.

Можно прослеживать распространение сигналов по схеме; это может быть удобно при отладке схем. Для этого нужно в меню «Моделировать» снять флажок «Моделирование включено», затем выбрать пункт «Сбросить моделирование», и далее последовательно выбирать пункт «Шаг моделирования», наблюдая распространение сигнала.

Изменение во времени значений на контактах и некоторых компонентах схемы можно автоматически записывать в виде таблицы в файл. Подробнее — в разделе «Запись в журнал» руководства пользователя Logisim.

# 3. Лабораторные работы

## 3.1. Лабораторная работа 1. Знакомство с программой Logisim. Комбинационные устройства. Минимизация

*Время на выполнение: 2 часа*

Для выполнения этой лабораторной работы необходимо внимательно прочитать и осмыслить главу 2 и параграфы 1.3 и 1.4 данного пособия.

**Задание 1.** Спроектировать в Logisim комбинационное устройство, имеющее несколько входов и *две группы* по несколько выходов. Количество входов и выходов, а также поведение устройства для каждого варианта приведены в таблице 3.1. Вариант определяется по последней цифре номера студента в списке группы.

Под «значением на входах» и «значением на выходах» понимается некоторое целое число, двоичное представление которого формируется отдельными входными или выходными битами. Последний бит в такой записи — младший (бит 0). Состояния на выходах каждой из двух групп зависят от состояний одних и тех же входов. Таким образом, для варианта 1, например, нужно реализовать 1 + 3 = 4 булевых функции от 5 переменных. Нумерация битов в многобитных значениях всегда начинается с нуля (с младшего бита). Однако нумерация обычных объектов (дней недели, букв, и т.п.) начинается с единицы. Если на входы поступает бессмысленное значение (например, 14 в качестве номера месяца), то значение на выходах может быть произвольным («don't care»).

Каждая из реализаций устройства должна быть составлена в Logisim на логических элементах в виде схемы с осмысленными названиями входов и выходов. Для отдельных битов многобитных значений названия должны оканчиваться на номер бита (например day2, day1, day0 (младший бит в конце) для варианта 4). Для составления таблиц истинности, их минимизации методом Куайна — Мак-Класки и генерации схемы используется модуль Logisim «Комбинационный анализ».

Для защиты работы нужно представить:

1. Таблицу истинности устройства.

2. Реализацию ДНФ, минимизированную методом Куайна — Мак-Класки.
3. Реализацию, предложенную Logic Friday, а в случае её чрезмерной громоздкости — собственную (полученную в результате применения тождеств булевой алгебры), содержащую значительно меньше элементов, чем реализация из пункта 2.

**Задание 2.** Повторить все пункты задания 1 с учётом следующей информации: ни при каких обстоятельствах на входы устройства не могут прийти одновременно все нули или все единицы. Это обстоятельство позволяет существенно упростить схему устройства (дополнительно минимизировать булевы функции), используя значения «don't care». Эта методика подробно описана в конце параграфа 1.4 данного пособия.

**Требования к выполнению работы:** все задания выполняются в одном файле проекта Logisim; каждое самостоятельное устройство должно быть оформлено в виде отдельной схемы с осмысленным названием всех входов и выходов и самой схемы. Для демонстрации работоспособности всех реализаций нужно составить в Logisim схему, содержащую в качестве подсхем все реализации устройства (их будет четыре), и одновременно подающую на их входы одно и то же тестовое значение, при этом значения на выходах различных реализаций должны совпадать. Для этого потребуется объединить с помощью компонента Разветвитель отдельные биты значений на входах и выходах подсхем в пучки проводов и показать многобитные значения на них, подключив несколько компонентов «Датчик», установив значение «Беззнаковое десятичное» для их атрибута «Основание». Так как старшие биты на входах и выходах подсхем расположены сверху, для всех разветвителей нужно из контекстного меню выбрать пункт «Расставить по убыванию».

**Дополнительные вопросы и задания:**

1. Насколько дополнительное обстоятельство из задания 2 упростило схемы с точки зрения количества элементов? А с точки зрения задержек (см. параграф 1.5)?
2. Позволила ли реализация, предложенная Logic Friday (или ваша собственная), сократить общую задержку устройства?
3. По какой формуле можно рассчитать общее количество ячеек в таблице истинности, зная количество входов и выходов комбинационного устройства?
4. Сравните своё устройство с устройствами из других вариантов: какие устройства сложнее в реализации — с большим количеством входов или большим количеством выходов?

5. Просмотрев задания всех вариантов, попробуйте предположить, какие из этих устройств действительно могут использоваться в цифровой технике, а какие приведены в задании лишь «для интереса».
6. Попробуйте реализовать своё устройство только на элементах И-НЕ. Стало ли оно от этого проще? Зачем это может понадобиться на практике?

*Таблица 3.1.* Варианты заданий для лабораторной работы 1

| **Вариант** | **Кол-во входов** | **Кол-во выходов** | **Поведение устройства** |
|---|---|---|---|
| **1** | 5 | 1 | Значение на входах — номер буквы в латинском алфавите. На выходе «1», если эта буква — согласная. В остальных случаях — «0». |
| | | 3 | Значение на выходах — номер самого старшего входа, несущего «1». |
| **2** | 4 | 3 | Значение на выходах — количество входов, несущих «0». |
| | | 2 | Значение на выходах — остаток от деления значения на входах на 3, увеличенный на 1. |
| **3** | 4 | 2 | Значение на входах — номер месяца. На выходах — количество слогов в названии этого месяца. |
| | | 3 | Значение на выходах — модуль разности количества нулей и единиц на входах. |
| **4** | 5 | 3 | Значение на входах — число в декабре 2012 года. На выходах — номер дня недели для этого числа. |
| | | 1 | На выходе «0», когда на входе больше нулей, чем единиц. В остальных случаях — «1». |
| **5** | 3 | 8 | Значение на выходах — факториал значения на входах, делённый на 21 и округлённый до целых. |
| | | 2 | Значение на выходах — номер самого младшего входа, несущего «0». |

*Окончание таблицы 3.1*

| **Вариант** | **Кол-во входов** | **Кол-во выходов** | **Поведение устройства** |
|---|---|---|---|
| **6** | 5 | 3 | Значение на выходах — количество входов, несущих «1», увеличенное на 2. |
| | | 1 | На выходе «1», когда на входах нет ни одной пары нулей, идущих подряд. В остальных случаях — «0». |
| **7** | 3 | 3 | Значение на входах — номер дня недели. На выходах — количество гласных в названии этого дня недели. |
| | | 7 | Когда на входах значение «7» — на всех выходах единицы. В любом другом случае единица только на одном выходе. Номер этого выхода задаётся значением на входах. |
| **8** | 5 | 2 | Значение на выходах — количество входов, несущих «0», не соседствующих с другими входами, несущими «0». |
| | | 2 | Значение на выходах — остаток от деления значения на входах на 3. |
| **9** | 5 | 1 | На выходе «0», когда на входе число, большее 21. В остальных случаях — «1». |
| | | 3 | Значение на выходах — количество цифр в записи римскими цифрами значения на входах. |
| **10** | 4 | 4 | Значение на выходах — остаток от деления значения на входах на 11. |
| | | 1 | На выходе «1», если значение на входах в двоичном представлении симметрично. В остальных случаях — «0». |

## 3.2. Лабораторная работа 2. Элементарные устройства памяти

*Время на выполнение: 2 часа*

Для временного хранения информации в цифровых схемах применяют различные устройства памяти. Из простейших устройств памяти мы рассмотрим три класса: триггеры, регистры и счётчики. Все эти устройства есть во встроенной библиотеке Logisim «Память». Они могут быть и асинхронными (см. параграф 1.6 пособия), но в рамках данного курса мы будем рассматривать только синхронные. У каждого синхронного устройства есть *синхронизирующий* (*тактовый*) вход. Любое изменение внутреннего состояния устройства происходит только в тот момент, когда уровень сигнала на тактовом входе меняется (в таком случае говорят, что тактовый вход *срабатывает*).

*Триггер* — простейшее устройство памяти (а значит, последовательностное устройство), хранящее один бит информации. Иными словами, триггер может иметь только два разных внутренних состояния — «0» или «1». В англоязычной литературе триггер называют «flip-flop». Существует четыре вида триггеров: D (data), T (toggle), JK (jump-kill) и RS (reset-set). Эти названия даны по названиям входов триггеров. Кроме этих входов каждый триггер имеет два выхода — Q (прямой) и $\overline{Q}$ (инверсный); значение на прямом выходе всегда совпадает со внутренним состоянием триггера, а значение на инверсном — противоположное. Каждый из четырёх типов триггеров имеет разное поведение. Таблицы истинности для них приведены в таблице 3.2. Q' означает значение, противоположное значению, хранимому в триггере в данный момент.

*Таблица 3.2.* Триггеры

| **D-триггер** | | **T-триггер** | | **JK-триггер** | | | **RS-триггер** | | |
|---|---|---|---|---|---|---|---|---|---|
| **D** | **Q** | **T** | **Q** | **J** | **K** | **Q** | **S** | **R** | **Q** |
| 0 | 0 | 0 | Q | 0 | 0 | Q | 0 | 0 | Q |
| 1 | 1 | 1 | Q' | 0 | 1 | 0 | 0 | 1 | 0 |
| | | | | 1 | 0 | 1 | 1 | 0 | 1 |
| | | | | 1 | 1 | Q' | 1 | 1 | ? |

Можно дать словесное описание поведения триггеров:

- **D-триггер:** когда тактовый вход срабатывает, значение, хранящееся в триггере, мгновенно становится значением входа D (данные).
- **T-триггер:** когда тактовый вход срабатывает, значение, хранящееся в триггере, меняется или остаётся прежним в зависимости от того, какое значение на входе T (переключение): «1» или «0».
- **JK-триггер:** когда тактовый вход срабатывает, значение, хранящееся в триггере, меняется, если на входах J и K единица; остаётся прежним, если на них 0; если значения на них различны, то значение становится единицей, если на входе J (прыжок) — «1»; или нулём, если на входе K (забой) — «1».
- **RS-триггер:** когда тактовый вход срабатывает, значение, хранящееся в триггере, остаётся неизменным, если на входах R и S — «0»; становится «0», если на входе R (сброс) — «1», и становится «1», если на входе S (установка) — «1». Поведение не определено, если на обоих входах «1». (В Logisim значение триггера остается неизменным.)

Физически триггеры реализуются на логических элементах (то есть в конечном итоге на транзисторах в составе интегральных схем), включенных, как правило, не совсем обычным для них способом — их выходы так или иначе соединяются с их входами. Как говорилось выше, триггеры могут быть синхронными и асинхронными. Тактовый вход триггеров и других устройств памяти в Logisim обозначается треугольником; в случае, когда вход нужно пометить буквой или строкой, используют «C» или «Clock». Синхронные триггеры, как правило, содержат большее количество логических элементов. Иногда синхронными ошибочно называют также особую разновидность триггеров, имеющих *разрешающий* вход. Изменение внутреннего состояния такого триггера происходит, когда на разрешающем входе «1». Этот вход иногда называют синхронизирующим, но на самом деле он таковым не является. На схемах ниже разрешающий вход отмечен буквой «E» (от англ. enable).

Асинхронные триггеры (с разрешающим входом или без него) иногда называют «прозрачными» (чаще в англоязычной литературе — “transparent”), а синхронные — «непрозрачными» (англ. “non-transparent” или “opaque”). Это связано с тем, что если на разрешающий вход (если таковой имеется) «прозрачного» триггера подать единицу, то помимо

записи в память триггера, входной сигнал будет непосредственно подаваться на выход триггера (то есть можно сказать, что триггер будет работать в качестве *повторителя*). Если при этом сигнал на входе зависит от сигнала на выходе (то есть выход и вход триггера связаны через внешнюю схему, не содержащую синхронных устройств), то схема начнёт возбуждаться. Иными словами, если в схеме есть своего рода «замкнутый круг», то чтобы предотвратить возбуждение схемы, нужно «разорвать» этот круг хотя бы в одном месте синхронным устройством (например, синхронным триггером). Об этом можно подробнее прочитать в параграфе 1.6.

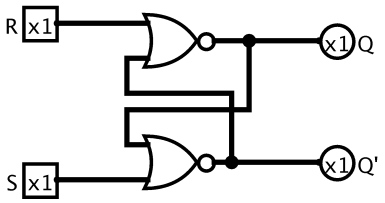


*Рис. 3.1.* Асинхронный RS-триггер

На схемах представлены различные реализации триггеров на логических элементах (рис. 3.1, рис. 3.2, рис. 3.3, рис. 3.4). На рисунке 3.4 показан D-триггер, использующий специальную «надстройку» над RS-триггером, делающую его синхронным. Однако существует общая техника построения синхронных триггеров из триггеров с разрешающим входом — *двухступенчатые триггеры* (англ. master-slave flip-flop). В таком триггере соединены два триггера, на разрешающие входы которых подаются противоположные значения. Двухступенчатый синхронный D-триггер показан на рисунке 3.5. При поступлении на его тактовый вход переднего фронта, обновится только состояние на выходе первой ступени, а обновление состояния всего триггера произойдёт при заднем фронте.

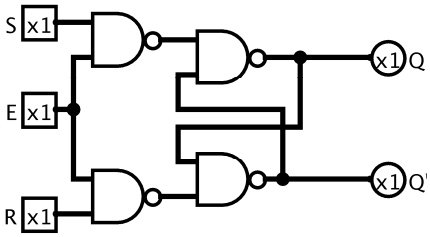


*Рис. 3.2. RS*-триггер с разрешающим входом

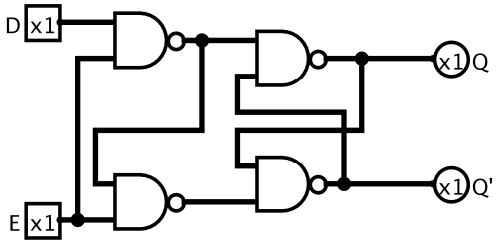


*Рис. 3.3.* D-триггер с разрешающим входом

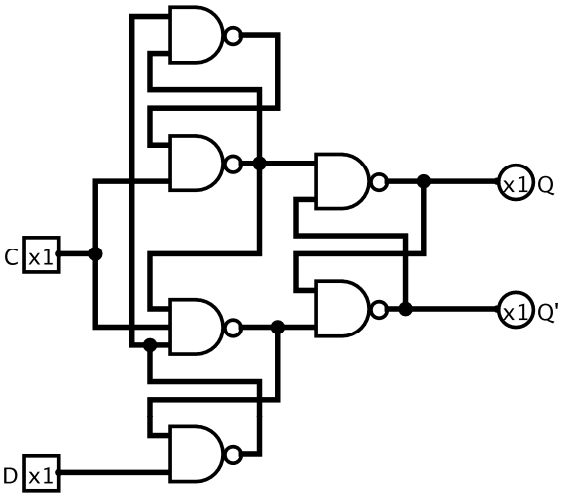


*Рис. 3.4.* Синхронный D-триггер

При моделировании триггеров на логических элементах в Logisim, следует установить флажок «Добавить шум к задержкам компонентов» в окне параметров

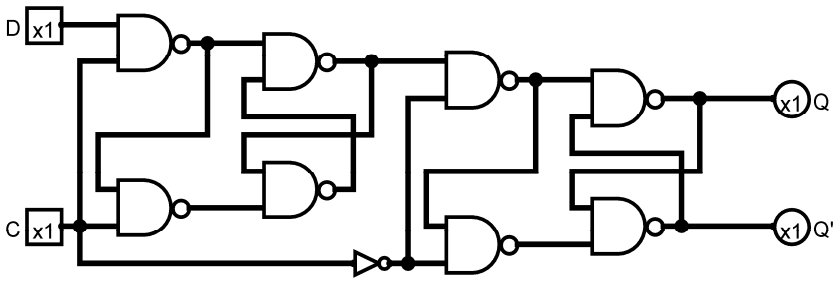



*Рис. 3.5.* Двухступенчатый синхронный D-триггер

проекта. Это нужно сделать для того, чтобы имитировать неравномерность реальных схем. Например, RS-триггер, построенный с использованием двух элементов ИЛИ-НЕ, будет возбуждаться без этой случайности, так как оба элемента будут обрабатывать свои входы «нога в ногу». До подачи первого значения на входы, состояние некоторых триггеров будет неопределённым, и Logisim будет показывать значение ошибки на выходах. Это особенности работы Logisim, на практике таких проблем не возникает.

*Регистр* — последовательностное логическое устройство, используемое для хранения n-разрядных (многобитных) двоичных *слов* (чисел) и выполнения преобразований над ними. Не считая счётчиков, в рамках данного курса мы будем использовать регистры только для хранения информации (компонент Logisim «Регистр»). Такой регистр имеет многобитный вход для загрузки данных, тактовый вход и многобитный выход, на который всегда поступает значение, сохранённое в регистре. В первом приближении можно представить себе синхронный регистр для хранения n-разрядных слов как группу из n синхронных D-триггеров с соединёнными тактовыми входами.

*Счётчик* — последовательностное логическое устройство, на выходы которого поступает двоичный код (многобитное значение), определяемый числом поступивших на его тактовый вход импульсов. Компонент Logisim «Счётчик» по сути является регистром, который меняет хранимое значение на единицу при поступлении очередного тактового импульса. Кроме того, у него имеются два дополнительных входа («загрузка» и «счёт»), которые позволяют выбирать его поведение — увеличивать значение, уменьшать его, или работать как обычный регистр.

В библиотеке Logisim «Память» есть компонент «*Генератор случайных чисел*». При срабатывании тактового входа он подаёт на выход очередное значение заданной разрядности из псевдослучайной последовательности.

Чтобы сделать в Logisim триггер, регистр или счётчик асинхронным, нужно установить для его атрибута «Срабатывание» значение «Высокий уровень». В таком случае тактовый вход будет работать как

разрешающий, и устройство станет «прозрачным».

Более подробную информацию о работе устройств памяти вообще и в Logisim в частности можно найти на страницах справки по библиотеке Logisim (библиотека «Память»).

**Задание 1.** Реализовать в Logisim асинхронный RS-триггер на логических элементах. Убедиться в том, что его поведение соответствует описанному в таблице и совпадает с поведением RS-триггера из встроенной библиотеки Logisim. Значения на входных контактах можно изменять инструментом «Нажатие».

**Задание 2.** Повторить задание 1 для RS-триггера с разрешающим входом.

**Задание 3.** Повторить задание 1 для D-триггера с разрешающим входом.

**Задание 4.** Повторить задание 1 для синхронного D-триггера.

**Задание 5.** Спроектировать асинхронный 8-разрядный регистр с разрешающим входом на основе D-триггера из задания 3.

**Задание 6.** Спроектировать синхронный 8-разрядный регистр на основе D-триггера из задания 4.

**Задание 7.** Спроектировать 4-разрядный счётчик на основе D-триггера из задания 4 и комбинационного устройства собственной разработки, имеющего 4 входа и 4 выхода. Это комбинационное устройство выдаёт на выходах 4-разрядное значение на единицу большее, чем 4-разрядное значение на его входах, а при подаче максимального значения выдаёт ноль.

**Требования к выполнению работы:** все задания выполняются в одном файле проекта Logisim; каждое самостоятельное устройство должно быть оформлено в виде отдельной схемы с осмысленным названием входов и выходов, а также самой схемы.

**Дополнительные вопросы и задания:**

1. Может ли существовать асинхронный D-триггер без разрешающего входа? Почему?
2. Чем D-триггер с разрешающим входом существенно отличается от синхронного D-триггера? Какую последовательность сигналов нужно подать на их входы, чтобы наглядно продемонстрировать это отличие?
3. Какое минимальное изменение нужно внести в JK-триггер, чтобы получить из него T-триггер?

## 3.3. Лабораторная работа 3. Декодер, мультиплексор и демультиплексор

*Время на выполнение: 2 часа*

*Декодер* — комбинационное логическое устройство, имеющее несколько выходов и многобитный управляющий вход. Если разрядность управляющего входа равна m, то количество выходов равно $2^m$. В каждый момент времени «1» будет поступать на один из выходов декодера, номер этого выхода будет равен многобитному значению, поступающему на управляющий вход. Иными словами происходит *декодирование* двоичного значения на входе в порядковый номер выхода. Декодер часто используется, когда нужно обратиться к какому-либо устройству по его коду (номеру); в дальнейшем мы не раз с этим встретимся. Чтобы построить декодер из логических элементов, нужно составить его таблицу истинности. Для примера рассмотрим построение декодера 2-к-4 (таблица 3.3). При условии, что разряды управляющего входа расставлены по убыванию, а номера выходов – по возрастанию, таблица истинности для декодера любой разрядности представляет собой единичную матрицу — по главной диагонали стоят единицы, а все остальные элементы равны нулю. Схема декодера представлена на рисунке 3.6.

*Таблица 3.3.* Таблица истинности декодера 2-к-4

| c1 | c0 | o0 | o1 | o2 | o3 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

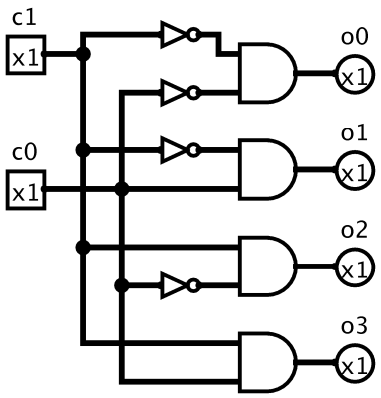



*Рис. 3.6.* Декодер 2-к-4

*Мультиплексор* — комбинационное логическое устройство, имеющее несколько информационных входов, многобитный управляющий вход и один выход. В каждый момент времени на выход подаётся сигнал с одного из информационных входов. Номер этого входа задаётся двоичным кодом, поступающим на управляющий вход. Если разрядность управляющего входа равна m, то количество информационных входов равно $2^m$. Реализация мультиплексора 4-к-1 представлена на рисунке 3.7. В его схему входит декодер 2-к-4.

На рисунке 3.8 показан ещё один вариант реализации мультиплексора — с использованием управляемых буферов.

Подробнее про управляемый буфер можно прочитать в статье «Управляемый буфер/инвертор» справки по библиотеке Logisim (библиотека «Элементы»). Вкратце — когда на управляющий вход управляемого буфера подаётся «1», он ведёт себя как буфер (то есть просто пропускает значение со своего входа на выход), а когда на управляющий вход подаётся «0», на выходе устанавливается высокоимпедансное состояние (неопределённость). Такой вариант реализации мультиплексора гораздо экономичнее, так как содержит меньшее количество элементов, однако при работе такой схемы должно быть гарантировано, что в каждый момент времени логическая единица будет поступать на управляющий вход строго одного буфера. В противном случае может произойти физическое повреждение устройства.

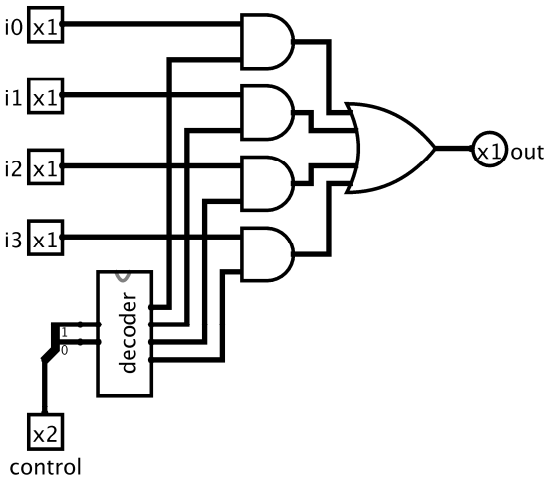



*Рис. 3.7.* Мультиплексор 4-к-1

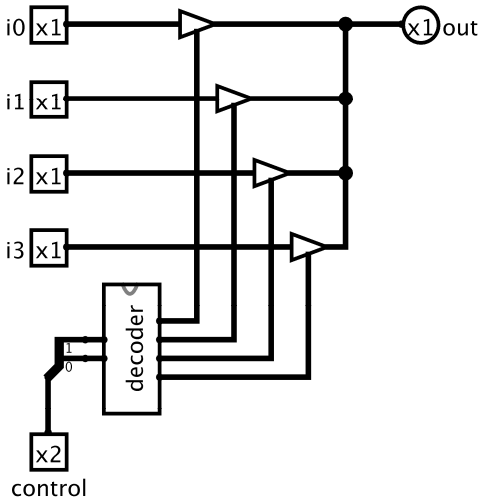



*Рис. 3.8.* Мультиплексор 4-к-1. Вариант с управляемыми буферами

Logisim позволяет задавать разрядность значения, проходящего через мультиплексор (то есть разрядность информационных входов и выхода). Можно представлять себе мультиплексор как аналог железнодорожной стрелки, управляемой управляющим входом. Мультиплексоры обозначают «MUX» (от англ. multiplexer), а также «MS» (от англ. multiplexer selector).

Мультиплексоры используются в цифровых схемах весьма часто. Одно из самых типичных применений — установка мультиплексора перед входом данных регистра. В этом случае на информационные входы мультиплексора подаются значения, которые могут быть записаны в регистр в разных ситуациях, а на управляющий вход — код (номер) значения, которое будет записано в текущем такте. Другое типичное применение — для преобразования нескольких параллельных сигналов в последовательность сигналов, проходящих по одному проводу. При таком использовании на управляющий вход циклически подаются

значения от 0 до $2^m - 1$, а на другом конце провода для обратного преобразования включается декодер или демультиплексор.

*Демультиплексор* — комбинационное логическое устройство, имеющее несколько выходов, многобитный управляющий вход и один информационный вход. В каждый момент времени на один из выходов подаётся сигнал с информационного входа. Номер выхода задаётся двоичным кодом, поступающим на управляющий вход. Если разрядность управляющего входа равна m, то количество выходов равно $2^m$. Реализация демультиплексора 1-к-4 представлена на рисунке 3.9. В его схему также входит декодер 2-к-4.

Logisim позволяет задавать разрядность значения, проходящего через мультиплексор. Демультиплексоры обозначают «DEMUX» или «DMX». Как уже говорилось выше, демультиплексор в паре с мультиплексором используется для передачи нескольких параллельных сигналов по одному проводу посредством их преобразования в последовательные сигналы. Ещё одно типичное применение демультиплексора — для составления сложных устройств памяти (например, регистрового файла) из более простых: на вход демультиплексора подаётся сигнал для записи, его выходы подключаются к входам данных регистров, а на управляющий вход подаётся номер регистра, в который должна производиться запись.

Информационные входы и выходы мультиплексора и демультиплексора в Logisim могут быть многобитными. Это означает, что такое устройство может одновременно перенаправлять группу битов. Допустим, что разрядность управляющего входа равна m, а разрядность информационных входов и выходов равна k. Тогда физически такой «многобитный» мультиплексор или демультиплексор — это группа из k однобитных мультиплексоров или демультиплексоров, и все их управляющие входы соединены в один, который и будет управляющим входом многобитного устройства. В случае мультиплексора каждый i-й одноразрядный мультиплексор будет иметь $2^m$ входов (как и всё многобитное устройство) и принимать на свой j-й вход i-й бит с j-го входа многоразрядного мультиплексора, а сигнал со своего выхода выдавать на i-й бит выхода многоразрядного мультиплексора. В случае демультиплексора каждый i-й

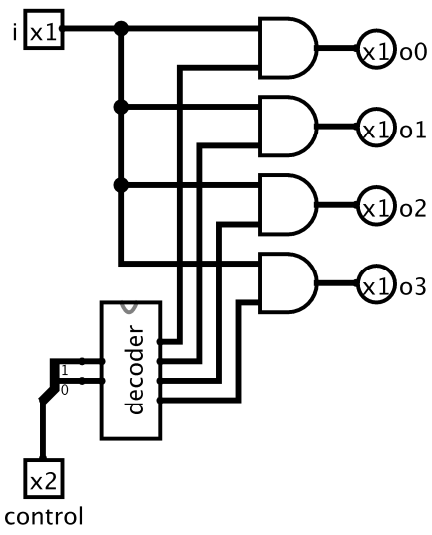


*Рис. 3.9.* Демультиплексор 1-к-4

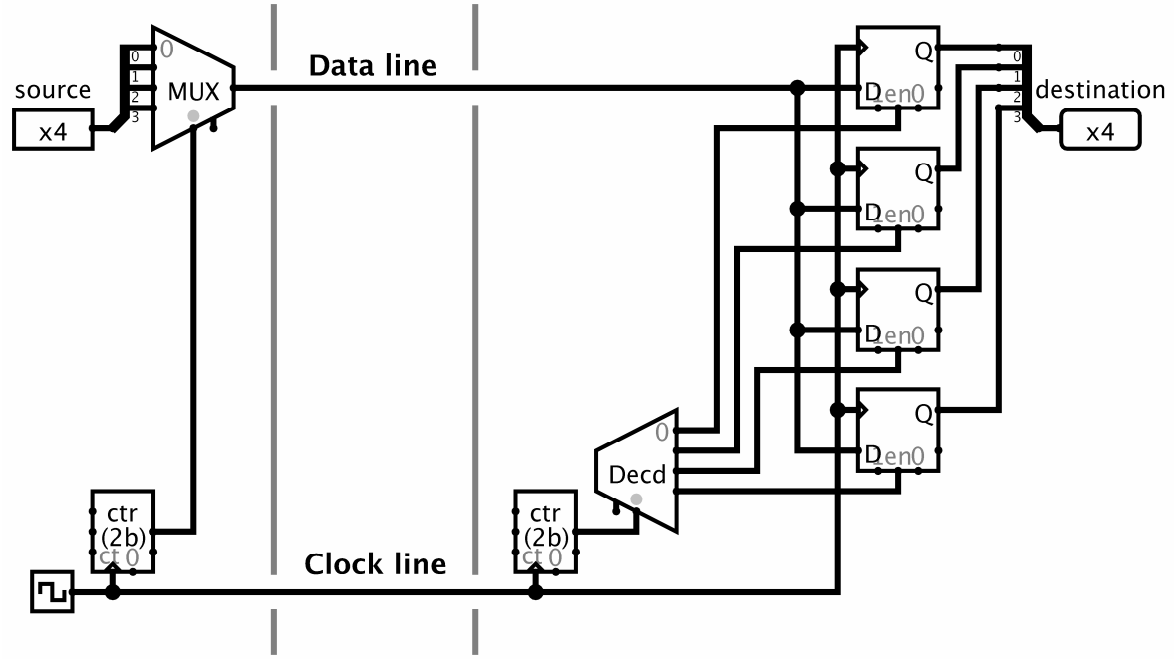



*Рис. 3.10.* Передача 4-битного значения посредством мультиплексирования

одноразрядный демультиплексор будет иметь $2^m$ выходов (как и всё многобитное устройство), принимать сигнал с i-го бита многоразрядного демультиплексора, и выдавать сигнал со своего j-го выхода на i-й бит j-го выхода многоразрядного демультиплексора.

Примитивная схема, демонстрирующая принцип передачи многоразрядного (4-разрядного в данном примере) значения по одному проводу приведена на рисунке 3.10. Такой приём называется *мультиплексированием*. Четырёхразрядное значение для передачи подаётся на вход «source». Счётчики в устройстве-передатчике и устройстве-приёмнике считают количество тактовых импульсов чтобы сформировать 2-битный управляющий сигнал для мультиплексора и декодера. Мультиплексор преобразует многоразрядное значение со входа в последовательные сигналы равной продолжительности, поступающие в каждый момент времени на входы всех D-триггеров. D-триггеры накапливают 4-битное значение, а декодер управляет тем, какой из четырёх триггеров должен сохранить новое значение с выхода демультиплексора. Принятое с линии и демультиплексированное значение снимается в выходов триггеров и поступает на выход «destination». После прохождения четырёх полных тактов на вход может быть подано новое значение, и ещё через четыре такта оно будет на выходах триггеров, и т.д. Понять принцип работы этой схемы будет гораздо проще после выполнения лабораторной работы 5.

Более подробную информацию о мультиплексоре, демультиплексоре и

декодере в Logisim можно найти на страницах справки по библиотеке Logisim (библиотека «Плексоры»).

**Задание 1.** Спроектировать одноразрядные мультиплексор и демультиплексор. Для мультиплексора реализовать оба варианта — с управляемыми буферами и без них. Разумеется, предварительно нужно спроектировать декодеры с соответствующей разрядностью управляющего входа. Разрядности управляющего входа мультиплексора и демультиплексора указаны в таблице 3.4. Вариант определяется по последней цифре номера студента в списке группы.

Для декодеров с разрядностью управляющего входа 4 и 5 количество выходов равно соответственно 16 и 32. Модуль «Комбинационный анализ» Logisim не позволяет создавать схемы с количеством выходов большим 12, поэтому декодер придётся делать двухступенчатым. Для «первой ступени» нужно построить с помощью модуля «Комбинационный анализ» декодер 3-к-8 с дополнительным однобитным разрешающим входом. Если на такой вход поступает «0», то и на всех выходах должен быть «0». «Вторая ступень» будет содержать 2 или 4 таких декодера, разветвитель и дополнительное комбинационное устройство. Разветвитель разделяет значение, поступающее на управляющий вход декодера 4-к-16 или 5-к-32 таким образом, что младшие 3 бита посылаются на управляющие входы декодеров 3-к-8, а оставшиеся старшие биты (1 или 2) поступают на входы дополнительного комбинационного устройства, выходы которого управляют разрешающими входами декодеров 3-к-8. Это комбинационное устройство само будет декодером (1-к-2 или 2-к-4).

**Задание 2.** Используя одноразрядные мультиплексор и демультиплексор из задания 1, спроектировать многоразрядные мультиплексор и демультиплексор. Разрядности данных указаны в таблице 3.4.

**Требования к выполнению работы:** все задания выполняются в одном файле проекта Logisim; каждое самостоятельное устройство должно быть оформлено в виде отдельной схемы с осмысленным названием всех входов и выходов и самой схемы. Входы мультиплексоров и выходы демультиплексоров должны быть расположены в порядке возрастания их номеров. Для защиты каждого задания нужно продемонстрировать работоспособность каждого отдельного устройства на какой-либо тестовой схеме, в которой нужно сравнить поведение спроектированных устройств с поведением соответствующих устройств из библиотеки Logisim (они должны совпадать).

**Дополнительные вопросы и задания:**

1. Составьте формулу для вычисления суммарного количества логических элементов в составе декодера, мультиплексора и демультиплексора в зависимости от разрядности управляющего входа и разрядности данных.
2. Оцените общую задержку распространения сигнала для декодера, мультиплексора и демультиплексора.
3. Между двумя городами проложена линия для передачи мультиплексированного сигнала, скорость передачи данных по ней — 1024 Килобит в секунду. Кабель на линии заменили многожильным, и теперь она способна передавать параллельно 32 бита, но максимальная частота передачи данных для нового кабеля в 2 раза ниже. Какой стала скорость передачи данных после замены кабеля?

*Таблица 3.4.* Варианты заданий для лабораторной работы 2

| Вариант | Разрядность управляющего входа | | Разрядность данных | |
|---|---|---|---|---|
| | Мульти-плексора | демульти-плексора | мульти-плексора | демульти-плексора |
| **1** | 4 | 5 | 6 | 7 |
| **2** | 5 | 3 | 8 | 8 |
| **3** | 3 | 5 | 5 | 6 |
| **4** | 5 | 3 | 7 | 5 |
| **5** | 5 | 4 | 8 | 7 |
| **6** | 4 | 5 | 5 | 6 |
| **7** | 5 | 4 | 6 | 5 |
| **8** | 5 | 3 | 8 | 7 |
| **9** | 3 | 5 | 7 | 8 |
| **10** | 5 | 3 | 6 | 5 |

## 3.4. Лабораторная работа 4. Арифметика. АЛУ

*Время на выполнение: 6 часов*

Группу битов можно рассматривать как отдельные разряды многобитного значения, которое, в свою очередь, можно рассматривать как двоичное представление целого числа. Мы уже не раз сталкивались с такой интерпретацией многоразрядных сигналов. В Logisim группу однобитных проводов можно объединить в *пучок*; в таком случае можно говорить, что пучок передаёт многобитное значение или целое число. При записи двоичного представления целого числа, нули и единицы записывают начиная со старшего разряда (бита), а при нумерации битов «нулевым» считают младший (последний в записи) бит.

Однако таким очевидным способом можно представлять только неотрицательные целые числа. Существует несколько способов представления отрицательных целых чисел с помощью многобитных двоичных значений: *прямой код*, *дополнительный код*, *дополнение до 1*, и некоторые другие. Самым удобным, а потому самым популярным способом представления является дополнительный код, который также называют *дополнением до 2* (англ. *two's complement*). Рассмотрим его подробно на примере восьмибитных значений.

При представлении неотрицательных чисел, восемью битами можно представить $2^8 = 256$ различных чисел: от 0 до 255 включительно. При использовании дополнительного кода, теми же восемью битами можно представить числа от -128 до 127, при этом ноль и положительные числа имеют в старшем разряде «0», а отрицательные — «1». Алгоритм нахождения противоположного по знаку значения для положительного числа в дополнительном коде показан с примерами в таблице 3.5.

*Таблица 3.5.* Алгоритм нахождения отрицания в дополнительном коде

| Действие | Пример 1 | Пример 2 |
|---|---|---|
| 1. Начиная справа, найти первую «1» | 0101001 (41) | 0101100 (44) |
| 2. Инвертировать все биты слева от неё | **1010111** (-41) | **1010**100 (-44) |

Крайне важно понимать, что дополнительный код — это не способ «превратить» положительное число в отрицательное, а лишь способ

*интерпретации* двоичного значения. Одну и ту же последовательность бит определённой длины можно интерпретировать как положительное число, если рассматривать её как беззнаковое представление, или как отрицательное число, если считать её числом, записанным в дополнительном коде. Чтобы чётче уяснить этот момент, рассмотрим обе интерпретации для нескольких восьмибитных чисел (таблица 3.6).

Главное преимущество дополнительного кода — многие арифметические операции над числами, представленными в дополнительном коде, осуществляются цифровыми устройствами так же, как и над беззнаковыми числами. Попробуем сложить двоичные числа 00000001 и 10000001, считая что это беззнаковое представление, то есть складываться будут десятичные числа 1 и 129. Двоичный результат сложения — 10000010, и если интерпретировать это значение как беззнаковое, то оно означает 130, что соответствует действительности. С другой стороны, если бы мы интерпретировали слагаемые как числа в дополнительном коде, то складывали бы 1 и -127. Результат 10000010 в дополнительном коде означает -126, что снова соответствует действительности (1 + (-127) = -126).

В Logisim есть встроенная библиотека «Арифметика», содержащая компоненты для выполнения арифметических операций над многобитными двоичными значениями. Вы можете задавать разрядность этих значений. Почти все компоненты из библиотеки «Арифметика» дают правильные результаты, если интерпретировать и операнды, и результат или как беззнаковые числа, или как числа в дополнительном коде. Исключение составляет компонент «Делитель» — он осуществляет беззнаковое деление. Компонент «Компаратор», сравнивающий два значения, имеет атрибут «Формат числа», который позволяет задать, как следует интерпретировать входные значения. Компонент

*Таблица 3.6.* Примеры представления чисел в дополнительном коде

| Двоичное значение | Представление в дополнительном коде | Беззнаковое представление |
|---|---|---|
| 00000000 | 0 | 0 |
| 00000001 | 1 | 1 |
| ... | ... | ... |
| 01111110 | 126 | 126 |
| 01111111 | 127 | 127 |
| 10000000 | -128 | 128 |
| 10000001 | -127 | 129 |
| 10000010 | -126 | 130 |
| ... | ... | ... |
| 11111110 | -2 | 254 |
| 11111111 | -1 | 255 |

«Отрицатель» позволяет находить противоположное по знаку число (отрицание) в дополнительном коде. Более подробно о каждом компоненте можно прочитать в справке по библиотеке Logisim «Арифметика».

Обратите внимание, что при преобразовании значения с меньшей разрядностью (например, восьмибитного) в значение с большей разрядностью (например, шестнадцатибитное), представленных в дополнительном коде, старшие биты нового значения (восемь бит в нашем случае) нужно заполнить нулями или единицами в зависимости от того, является число положительным или отрицательным. Для этой цели можно использовать компонент Logisim «Расширитель битов» (библиотека «Проводка»), со значением «Знак» для атрибута «Тип расширения».

В библиотеке Logisim «Проводка» есть компонент «Датчик», который отображает многобитные значения с возможностью выбирать представление значения: двоичное, восьмеричное, беззнаковое десятичное, знаковое десятичное (дополнительный код), и шестнадцатеричное. Это крайне удобный компонент при отладке схем, связанных с арифметикой.

Все компоненты, выполняющие арифметические операции (сумматор, вычитатель, множитель, делитель, и т.д.) — комбинационные логические устройства, а значит, могут быть реализованы на логических элементах. Рассмотрим принципы их построения.

Начнём с сумматора. Вспомним, как осуществляется сложение «столбиком»:

```
  ..
  1234
+ 5678
  ====
  6912
```

Точки стоят над разрядами, которые принимают единицу от предыдущего (младшего) разряда, так как сумма цифр предыдущего разряда больше 9. Правила сложения одинаковы для любой позиционной системы счисления, поэтому сложение «столбиком» в двоичной системе будет выглядеть точно также:

```
  .. ...
   11011100 (220)
+  01001110  (78)
   ========  ===
  100101010 (298)
```

В отличие от десятичной системы, в двоичной системе при сложении

цифр одного разряда могут получиться только числа 0, 1, 10 (2), 11 (3). В соответствующий разряд результата попадает младшая цифра этого числа, а старшая цифра передаётся для сложения со следующим разрядом. Эта старшая цифра называется *битом переноса* (англ. *carry bit*). Обратите внимание, что в результате сложения в данном примере получается двоичное значение длиной больше 8 битов. Однако выход сумматора с восьмибитными входами тоже восьмибитный, поэтому дополнительный старший бит не попадёт в результат сложения, и результат, вообще говоря, будет неверным. Но бит переноса из самого старшего разряда восьмибитного значения будет единицей, и этот бит, как правило, подаётся на специальный выход сумматора, и при необходимости может быть учтён при разработке схемы.

Получается, что элементарное устройство для выполнения сложения над одним разрядом должно принимать на вход две цифры из слагаемых (A и B) и бит переноса из предыдущего разряда (Cin — carry in), а на выходе давать цифру для соответствующего разряда результата (S — sum) и бит переноса в следующий разряд (Cout — carry out). Это устройство называется *полным одноразрядным сумматором* (англ. *1-bit full adder*). Составив таблицу истинности для этих двух булевых функций от трёх переменных, получим реализацию полного сумматора (рис. 3.11).

Следующий шаг — построить на основе нескольких одноразрядных сумматоров многоразрядный сумматор, способный складывать многобитные значения. Самое очевидное решение — подавать бит переноса с выхода Cout каждого полного одноразрядного сумматора на вход Cin сумматора следующего разряда. Получившееся в результате устройство — *многоразрядный сумматор с последовательным переносом* (англ. ripple carry adder), его блок-схема показана на рисунке 3.12. Такой многоразрядный сумматор прост в реализации, но он очень медленный: каждый одноразрядный сумматор должен дождаться бита переноса из предыдущего разряда, то есть сумматоры соединены последовательно, а значит, задержки элементов одноразрядных сумматоров суммируются. В результате общая задержка многоразрядного сумматора получается огромной.

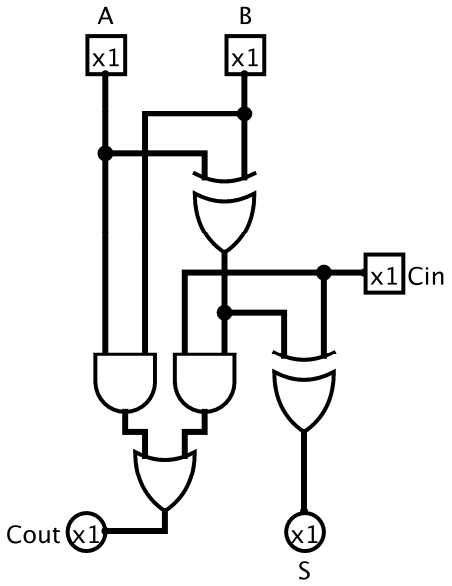



*Рис. 3.11.* Полный одноразрядный сумматор

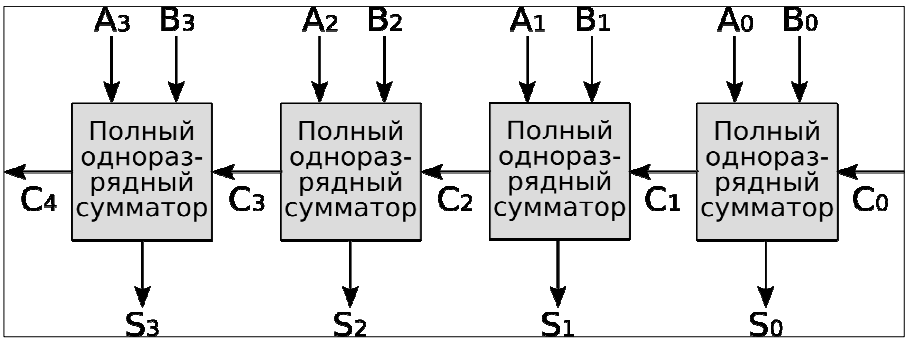



*Рис. 3.12.* Четырёхразрядный сумматор с последовательным переносом

Существует многоразрядный сумматор с параллельным переносом (англ. carry look-ahead adder), называемый также схемой ускоренного переноса, обладающий значительно меньшей общей задержкой. Полные одноразрядные сумматоры в составе схемы ускоренного переноса имеют небольшое отличие от одноразрядных сумматоров, описанных выше — вместо того, чтобы подавать на выход один бит переноса, они выдают два бита — G (generate) и P (propagate). Бит G — единица, если данный разряд генерирует бит переноса, и может быть вычислен как $G_i = A_i \cdot B_i$. Здесь и далее латинские буквы с индексами означают однобитные значения, индексы — номера разрядов, а знаки умножения и сложения обозначают логическое умножение и сложение (т. е. операции И и ИЛИ). Бит P — единица, если данный разряд распространяет (пропускает в старший разряд) бит переноса, и может быть вычислен как $P_i = A_i \oplus B_i$ (знак $\oplus$ означает исключающее ИЛИ), что с учётом некоторых особенностей схемы ускоренного переноса может быть заменено выражением $P_i = A_i + B_i$.

Значение бита переноса для разряда может быть вычислено из значений битов G, P и бита переноса предыдущего разряда следующим образом: $C_{i+1} = G_i + P_i \cdot C_i$. Зная это, попробуем спроектировать четырёхразрядную схему быстрого переноса. Запишем выражения для битов переноса каждого разряда:

$C_1 = G_0 + P_0 \cdot C_0$
$C_2 = G_1 + P_1 \cdot C_1$
$C_3 = G_2 + P_2 \cdot C_2$
$C_4 = G_3 + P_3 \cdot C_3$

Подставляя $C_1$ в $C_2$, затем $C_2$ в $C_3$, а затем $C_3$ в $C_4$, получим расширенные выражения:

$C_1 = G_0 + P_0 \cdot C_0$
$C_2 = G_1 + G_0 \cdot P_1 + C_0 \cdot P_0 \cdot P_1$
$C_3 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + C_0 \cdot P_0 \cdot P_1 \cdot P_2$
$C_4 = G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 \cdot P_3 + G_0 \cdot P_1 \cdot P_2 \cdot P_3 + C_0 \cdot P_0 \cdot P_1 \cdot P_2 \cdot P_3$

Обратите внимание, что теперь ни один бит переноса не зависит от предыдущего (кроме $C_0$, который подаётся в схему извне), а значит, все они вычисляются параллельно, что очень существенно сокращает общую задержку схемы. Общая блок-схема четырёхразрядного сумматора с параллельным переносом приведена на рисунке 3.13.

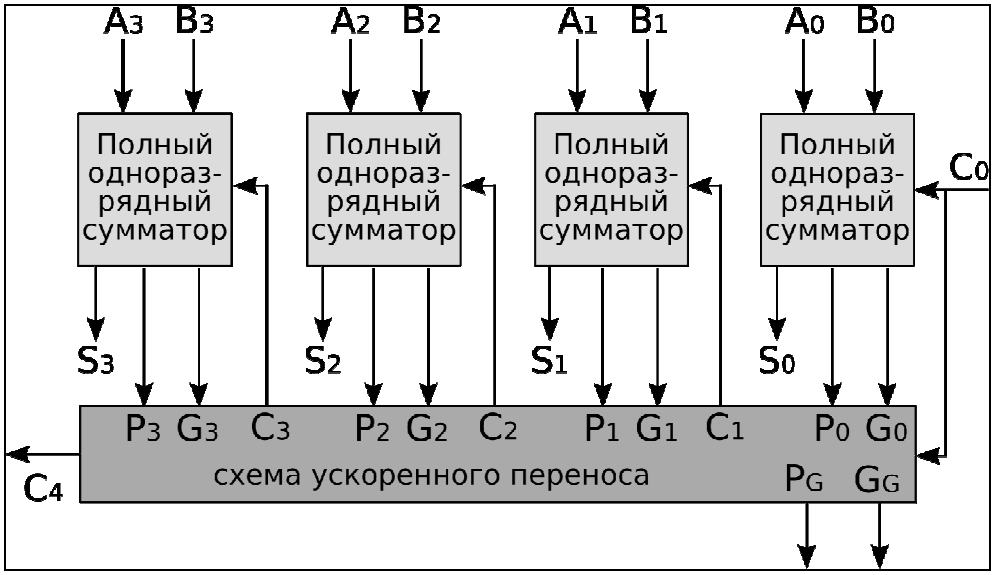



*Рис. 3.13.* Четырёхразрядный сумматор с параллельным переносом

Используя тот же метод, можно объединить четыре четырёхразрядных сумматора в шестнадцатиразрядный (рис. 3.14). Биты $G_G$ и $P_G$, которые схемы ускоренного переноса будут посылать в вышестоящую схему, вычисляются по формулам:

$$P_G = P_0 \cdot P_1 \cdot P_2 \cdot P_3$$

$$G_G = G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 \cdot P_3 + G_0 \cdot P_1 \cdot P_2 \cdot P_3$$

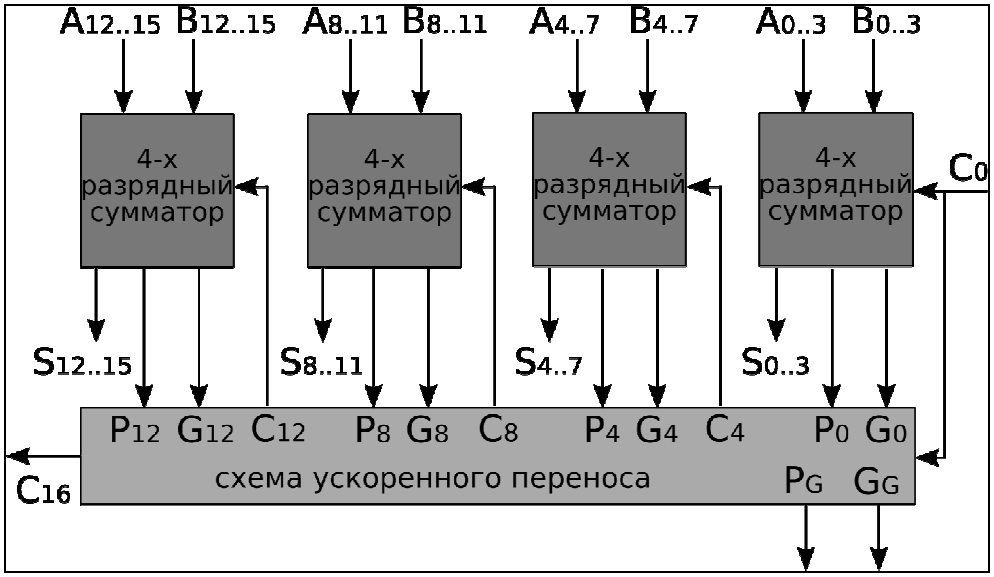



*Рис. 3.14.* Шестнадцатиразрядный сумматор с параллельным переносом

Несмотря на то, что на этих блок-схемах схемы ускоренного переноса принимают на входах биты с разными индексами, их внутреннее устройство полностью идентично и реализует шесть приведённых выше формул.

С помощью Logisim можно прослеживать распространение сигнала по схеме; это полезно сделать, чтобы сравнить общую задержку сумматоров с последовательным и параллельным переносом. Для этого нужно в меню «Моделировать» снять флажок «Моделирование включено», затем выбрать пункт «Сбросить моделирование», и далее последовательно выбирать пункт «Шаг моделирования», наблюдая распространение сигнала.

Пользуясь приведённым здесь математическим аппаратом, можно спроектировать многоразрядный сумматор с параллельным переносом любой разрядности непосредственно (без иерархической структуры), однако сложность устройства с увеличением разрядности будет расти очень быстро.

Существует два подхода к проектированию устройства для вычитания (*вычитателя*, англ. subtractor). Первый состоит в том, чтобы спроектировать аналог полного одноразрядного сумматора для вычитания и использовать его в многоразрядных вычитателях. Такое устройство (*полный одноразрядный вычитатель*) будет передавать старшим разрядам не бит переноса, а *бит займа* (англ. borrow bit). Схема полного одноразрядного вычитателя показана на рисунке 3.15. Его входы: X — уменьшаемое, Y — вычитаемое, Bin (borrow in) — вход займа; выходы: D (difference) — разность, Bout (borrow out) — выход займа. Трёхвходовый элемент исключающее ИЛИ выдаёт единицу, когда на нечётном количестве входов единица. Построение многоразрядного вычитателя с последовательным займом ничем не отличается от построения аналогичного сумматора. Такой вычитатель будет работать и с беззнаковой интерпретацией чисел и с дополнительным кодом.

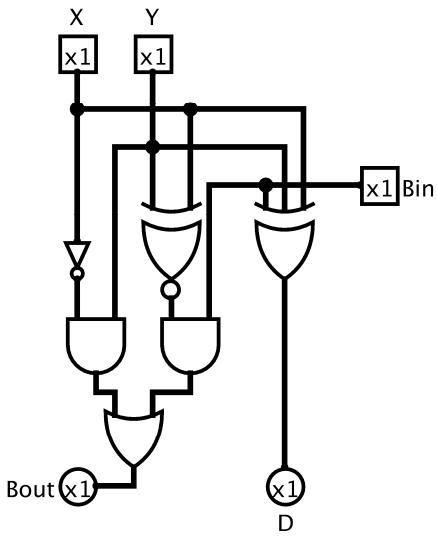



*Рис. 3.15.* Полный одноразрядный вычитатель

Второй подход — воспользоваться тем фактом, что вычитание — это сложение с числом, противоположным по знаку вычитаемому (X - Y = X + (-Y)). В вычитателе, построенном по такому принципу, сначала для вычитаемого будет находиться противоположное по знаку

значение (в дополнительном коде; этот алгоритм описан выше), а затем будет производиться сложение с помощью любого многоразрядного сумматора.

При построении устройства для умножения чисел (*множителя*) в первую очередь нужно помнить, что разрядность результата в два раза больше, чем разрядность сомножителей. Обычная практика — выдавать старшую и младшую половины результата на разные многобитные выходы, так что каждый из них имеет разрядность исходных данных (сомножителей). Построение множителя — задача весьма сложная; мы рассмотрим только самый простой вариант реализации, он использовался в самых первых компьютерах. Для этого вспомним умножение «столбиком»:

```
   123
 x 456
 ======
   738  (123·6)
  615   (123·5, сдвинутое на одну позицию влево)
+ 492    (123·4, сдвинутое на две позиции влево)
 ======
 56088
```

Умножение в двоичной системе будет производиться тем же способом:

```
    1110  (14 в двоичном представлении)
  x 1011  (11 в двоичном представлении)
    ====
    1110  (1110·1)
   1110   (1110·1, сдвинутое на одну позицию влево)
  0000    (1110·0, сдвинутое на две позиции влево)
+ 1110    (1110·1, сдвинутое на три позиции влево)
=========
 10011010   (154 в двоичном представлении)
```

Иными словами нужно только умножать (а умножение сводится либо к копированию первого множителя, либо к заполнению всех разрядов нулями), сдвигать полученные произведения и последовательно складывать их. Это можно делать потактово (тогда множитель перестанет быть комбинационным устройством), или реализовав все операции в виде одного комбинационного устройства (оно будет иметь очень большую общую задержку, так как сумматоры будут включены последовательно). Поскольку левые сдвиги осуществляются на заранее известные количества позиций, то их реализация даже не будет требовать логических элементов — лишь копирование значений со входов множителя на входы сумматоров со смещением. Поскольку у нас

уже есть готовые сумматоры, реализация множителя по описанному алгоритму не составит труда. Обратите внимание, что разрядность сумматоров, как и разрядность результата, в два раза больше разрядности исходных сомножителей.

Множитель, реализующий такой принцип, будет правильно работать только для беззнаковых чисел; если интерпретировать сомножители и результат как числа в дополнительном коде, результат будет неверным при хотя бы одном отрицательном сомножителе.

Реализация устройства для целочисленного деления (с остатком) является весьма сложной задачей; мы не будем рассматривать её в этом курсе. Напомним только, что как и в случае с умножением, результатом деления двух чисел разрядности n будут *два* числа разрядности n — частное и остаток; как правило, они выводятся на разные выходы.

Устройства для арифметических операций, рассмотренные выше, довольно редко используются в схемах непосредственно. Как правило, они включаются в состав *арифметико-логического устройства* (АЛУ). АЛУ — блок процессора, который служит для выполнения арифметических и логических преобразований над данными, представляемыми в виде многоразрядных значений (машинных слов), называемых операндами. В большинстве случаев АЛУ — комбинационное устройство. Примитивное АЛУ имеет два входа для операндов, один или два (если реализованы умножение и/или деление с остатком) выходов для результата, а также управляющий вход, на который подаётся код операции, которую необходимо выполнить над операндами. Опционально АЛУ может принимать и выдавать бит переноса, выдавать сигнал переполнения и деления на ноль, и т.д. Типичное условно-графическое обозначение АЛУ приведено на рисунке 3.16. Здесь A и B — операнды, R — результат, O — код операции.

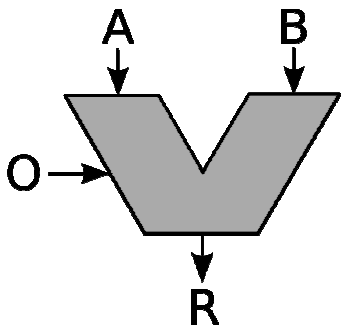


*Рис. 3.16.* Условное графическое обозначение АЛУ

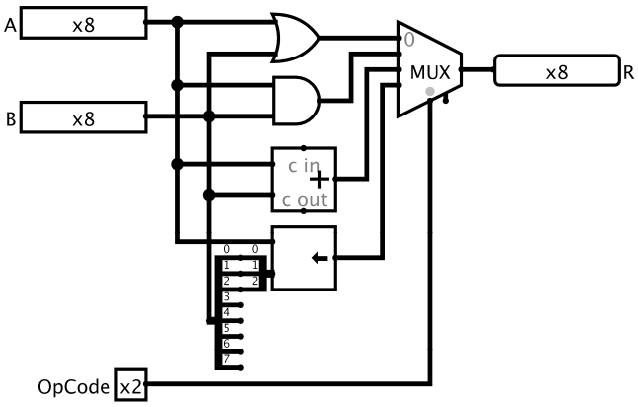


*Рис. 3.17.* Примитивное восьмибитное АЛУ

На рисунке 3.17 изображена схема примитивнейшего восьмибитного АЛУ. Оно может выполнять четыре операции: побитовое ИЛИ, побитовое И, сложение и логический левый сдвиг. Двухбитный код операции

поступает на управляющий вход мультиплексора, и на выход поступает результат выбранной операции. Левый сдвиг требует в качестве второго операнда трёхбитное значение, поэтому из операнда B предварительно выделены три младших бита с помощью компонентов «Разветвитель».

В этой лабораторной работе мы рассматривали арифметические операции только над целыми числами. Естественно, существуют цифровые устройства для выполнения операций над действительными числами (числами с плавающей точкой). Способы представления таких чисел в двоичном коде и организация устройств для работы с ними — отдельная и весьма сложная тема, она выходит за рамки этого курса. Вы можете рассмотреть её самостоятельно и даже попробовать реализовать такие устройства в Logisim.

**Задание 1.** Спроектировать в Logisim полный одноразрядный сумматор.

**Задание 2.** Спроектировать 8-разрядный сумматор с последовательным переносом, используя одноразрядный сумматор из задания 1.

**Задание 3.** Спроектировать 4-разрядный сумматор с параллельным переносом, используя видоизменённый полный одноразрядный сумматор.

**Задание 4.** Спроектировать 8-разрядный сумматор с параллельным переносом, используя два 4-разрядных сумматора из задания 3.

**Задание 5.** Спроектировать 16-разрядный сумматор с параллельным переносом, используя четыре 4-разрядных сумматора из задания 3.

**Задание 6.** Спроектировать полный одноразрядный вычитатель.

**Задание 7.** Спроектировать 8-разрядный вычитатель с последовательным займом, используя одноразрядный вычитатель из задания 6.

**Задание 8.** Спроектировать 8-разрядное устройство для нахождения числа, противоположного по знаку данному (отрицатель) в дополнительном коде. Сделать это можно, составив таблицу истинности вручную, однако такой способ весьма трудоёмок. Гораздо проще получить таблицу истинности, проанализировав с помощью модуля «Комбинационный анализ» схему, содержащую компонент «Отрицатель» из встроенной библиотеки Logisim.

**Задание 9.** Спроектировать 8-разрядный вычитатель, используя сумматор из задания 4 и отрицатель из задания 8.

**Задание 10.** Спроектировать множитель для беззнаковых чисел с 8-разрядными входами и двумя 8-разрядными выходами (для старшей и

младшей половин результата) в виде единого комбинационного устройства. Сумматоры в составе множителя — 16-разрядные с параллельным переносом из задания 5.

**Задание 11.** Спроектировать АЛУ для выполнения операций над двумя 8-разрядными операндами. АЛУ должно поддерживать выполнение восьми операций: сложение, вычитание, нахождение отрицания для первого операнда, умножение, поразрядное И, поразрядное ИЛИ, поразрядное исключающее ИЛИ, поразрядное НЕ над первым операндом. Все арифметические операции кроме умножения должны правильно работать как для беззнаковых чисел, так и для чисел в дополнительном коде. Вход кода операции — 3-разрядный. Соответствие кодов и операций — на ваше усмотрение.

Младшая половина результата умножения должна подаваться на общий выход для результата, а старшая половина — на специальный выход. Для выполнения поразрядных логических операций нужно предварительно спроектировать устройства, состоящие только из логических элементов с одноразрядными входами. Для выполнения арифметических операций должны использоваться только устройства, спроектированные в предыдущих заданиях лабораторной работы; какие из реализаций выбирать — на ваше усмотрение.

Мультиплексор в составе АЛУ — с восемью 8-разрядными информационными входами, реализованный на логических элементах с одноразрядными входами в виде отдельной подсхемы. Его реализация рассматривалась в лабораторной работе 3.

Убедитесь, что кроме контактов, проводов, разветвителей, констант и датчиков, все подсхемы АЛУ на всех уровнях иерархии содержат *только* логические элементы с одноразрядными входами. Для этого воспользуйтесь возможностью «Получить статистику схемы» из меню «Проект» Logisim. Какое общее количество логических элементов использовано в вашей реализации АЛУ?

**Требования к выполнению работы:** все задания выполняются в одном файле проекта Logisim; каждое самостоятельное устройство (начиная от одноразрядного сумматора и кончая АЛУ) должно быть оформлено в виде отдельной схемы с осмысленным названием всех входов и выходов и самой схемы. Для защиты каждого задания нужно продемонстрировать работоспособность каждого отдельного устройства на нескольких примерах входных значений, расположив устройство на отдельной тестовой схеме, общей для всей работы. Эти тестовые входные значения должны задаваться компонентами «Константа», к каждому из которых (а также к выходным контактам) должен быть

подключен компонент «Датчик», настроенный на отображение знакового десятичного представления значений.

**Дополнительные вопросы и задания:**

1. Возможно ли уменьшить количество логических элементов в составе вашего АЛУ без потери функциональности? Попробуйте это сделать. Для этого можно попробовать выбрать другую реализацию отдельного устройства, совместить сумматор и вычитатель, минимизировать некоторые булевы функции.
2. Определите, у кого в группе наименьшее количество логических элементов в составе АЛУ. Почему?
3. Попробуйте оценить максимальную задержку вашего АЛУ при выполнении операции. Это можно сделать, вручную посчитав длину самой большой цепочки последовательно соединённых логических элементов. Попробуйте проследить распространение сигнала по подсхемам, воспользовавшись возможностью Logisim, доступной через пункт «Шаг моделирования» из меню «Моделировать».
4. Определите, у кого в группе минимальная общая задержка при выполнении операции АЛУ. Какие из реализаций арифметических устройств им были выбраны?
5. Являются ли «победители соревнований» из пунктов 2 и 4 одним и тем же человеком? Почему? Попробуйте ответить на этот вопрос, даже если «соревнования» не проводились.
6. Попробуйте найти в литературе и реализовать устройство для умножения с меньшей общей задержкой.
7. Попробуйте предложить реализацию устройства для целочисленного деления (это сложное задание).

Спроектированное в процессе выполнения работы АЛУ будет содержать несколько сотен логических элементов, возможно даже несколько тысяч; кроме того, оно будет иметь довольно существенную максимальную общую задержку, так как умножитель содержит весьма большое количество элементов, соединённых последовательно. Причина этих недостатков в том, что мы не задумывались о дополнительной минимизации и оптимизации схем. Можно было провести минимизацию после проектирования, кроме того, можно было задуматься о том, как существенно упростить АЛУ без потери функциональности, проведя тщательный анализ всех схем в отдельности, их соединения, и всего устройство в целом ещё на начальной стадии проектирования, то есть до воплощения в виде логических элементов.

В конце 1960-х годов в составе уже упоминавшейся серии микросхем 7400 в виде отдельной микросхемы было выпущено 4-битное АЛУ 74181 (отечественный аналог — 155ИПЗ). Оно позволяет выполнять все семь логических операций, перечисленных в таблице 1.1, сложение и вычитание операндов, а также некоторые другие комбинации логических и арифметических операций. Умножение и деление в этом АЛУ не реализованы из-за их большой сложности. Объединив несколько таких АЛУ, выполняющих операции над 4-битными операндами, можно получить АЛУ большей разрядности. Биты переноса и займа можно передавать от одного 4-битного АЛУ к другому как непосредственно (получая сумматор и вычитатель с последовательным переносом), так и через дополнительную схему ускоренного переноса (74181 также генерирует биты P и G). Схема АЛУ 74181 приведена на рисунке 3.18.

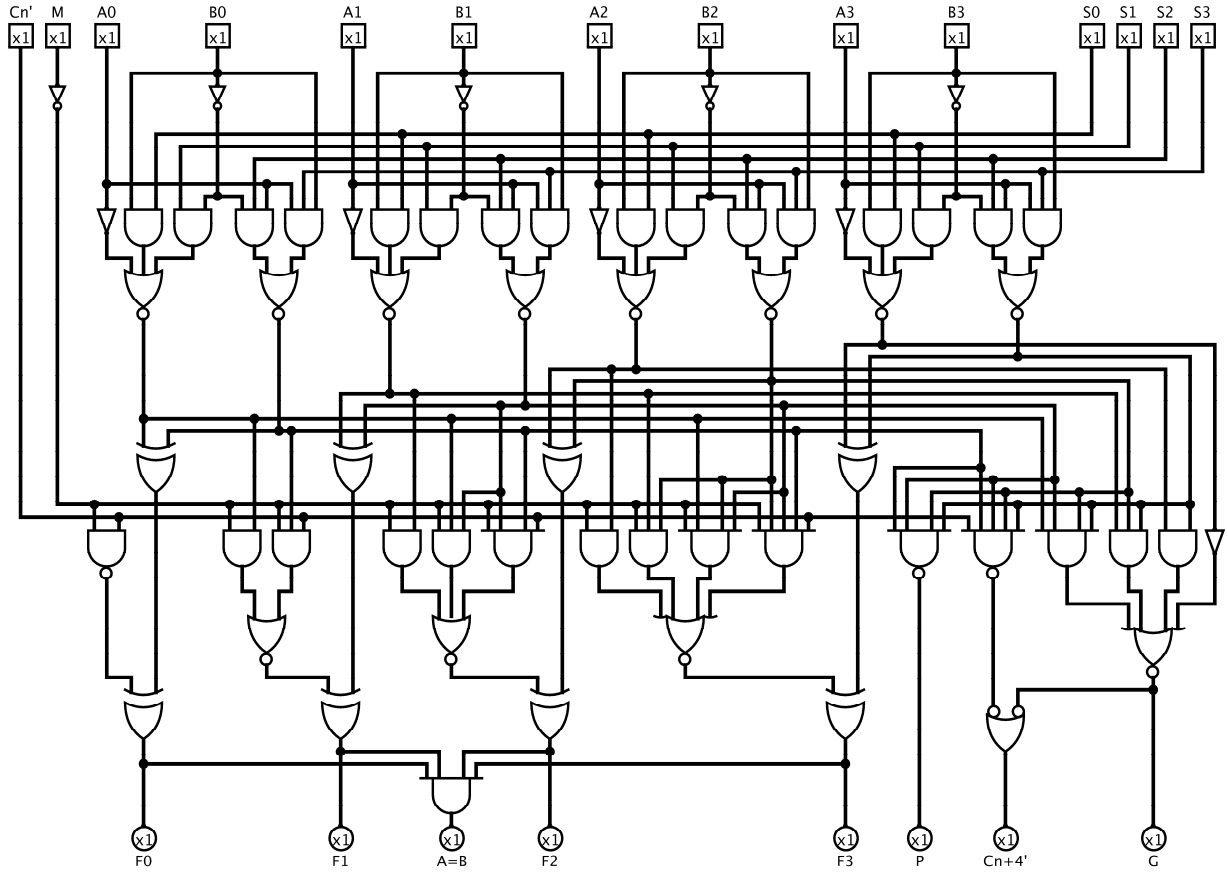



*Рис. 3.18.* АЛУ 74181

Схема этого АЛУ — великолепный пример того, насколько существенно можно уменьшить сложность внутреннего устройства при тщательнейшем анализе и квалифицированном проектировании: оно содержит всего около семидесяти логических элементов, а объединив два таких устройства, можно получить АЛУ эквивалентное по

функциональности спроектированному нами (и даже немного превосходящее его, за исключением операции умножения).

На этой схеме входы $A_0…A_3$ и $B_0…B_3$ — операнды, $S_0…S_3$ — код операции, $C_n'$ — бит переноса (или займа), поступающий из предыдущего разряда, M — признак того, что код операции задаёт логические операции (когда на этом входе 0, выполняются арифметические). Выходы $F_0…F_3$ — результат выполнения операции, A=B — признак равенства операндов при выполнении вычитания, $C_{n+4}'$ — выход бита переноса (или займа) в следующий разряд, P и G — биты propagate и generate для схемы ускоренного переноса. Вход $C_n'$ и выход $C_{n+4}'$ — инвертированные, то есть наличие переноса (или займа) обозначается логическим нулём.

Соответствие кодов и операций (логических и арифметических) приведено в таблице 3.7.

Словами «plus» и «minus» обозначены соответствующие арифметические операции чтобы избежать путаницы с логическим сложением (то есть с операцией ИЛИ). Символ "<<" означает логический левый сдвиг.

При M = 0 (то есть при выполнении арифметических операций) для всех операций plus и minus учитывается и генерируется бит переноса (или займа), а также биты P и G; при M = 0 выполняются исключительно поразрядные операции, то есть биты переноса не учитываются и не генерируются.

*Таблица 3.7.* Операции АЛУ 74181

| $S_3$ | $S_2$ | $S_1$ | $S_0$ | **Логические (M = 1)** | **Арифметические (M = 0, $C_n' = 1$)** |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $\overline{A}$ | A |
| 0 | 0 | 0 | 1 | $\overline{A + B}$ | A + B |
| 0 | 0 | 1 | 0 | $\overline{A} \cdot B$ | $A + \overline{B}$ |
| 0 | 0 | 1 | 1 | 0 | minus 1 |
| 0 | 1 | 0 | 0 | $\overline{A \cdot B}$ | A plus $A \cdot \overline{B}$ |
| 0 | 1 | 0 | 1 | $\overline{B}$ | (A + B) plus $A \cdot \overline{B}$ |
| 0 | 1 | 1 | 0 | $A \oplus B$ | A minus B minus 1 |
| 0 | 1 | 1 | 1 | $A \cdot \overline{B}$ | $A \cdot B$ minus 1 |
| 1 | 0 | 0 | 0 | $\overline{A} + B$ | A plus $A \cdot B$ |
| 1 | 0 | 0 | 1 | $\overline{A \oplus B}$ | A plus B |
| 1 | 0 | 1 | 0 | B | $(A + \overline{B})$ plus $A \cdot B$ |
| 1 | 0 | 1 | 1 | $A \cdot B$ | $A \cdot B$ minus 1 |
| 1 | 1 | 0 | 0 | 1 | A plus (A << 1) |
| 1 | 1 | 0 | 1 | $A + \overline{B}$ | (A + B) plus A |
| 1 | 1 | 1 | 0 | A + B | $(A + \overline{B})$ plus A |
| 1 | 1 | 1 | 1 | A | A minus 1 |

## 3.5. Лабораторная работа 5. Тактовые импульсы. Шина

*Время на выполнение: 4 часа*

В предыдущих лабораторных работах мы не пользовались сложными устройствами библиотеки компонентов Logisim, мы фактически проектировали их сами из логических элементов. Это было оправданно, так как мы изучали назначение различных цифровых компонентов, принципы их работы и внутреннее устройство. По сути, все предыдущие работы — лишь обязательная подготовка к собственно проектированию цифровых устройств, имеющих практическое применение. Начиная с этой лабораторной работы, мы непосредственно приступаем к такому проектированию, поэтому теперь, для экономии сил и времени, любой компонент, нужный для реализации задачи, мы будем брать из библиотеки компонентов Logisim — ведь теперь мы знаем, как он работает, и как при необходимости реализовать его на логических элементах.

Вопросы, связанные с понятием *тактовых импульсов* весьма подробно рассмотрены в параграфе 1.6 данного пособия; его прочтение необходимо для выполнения этой лабораторной работы.

Чтобы понять, что такое *шина*, рассмотрим небольшой практический пример. Допустим, у нас есть четыре устройства, которые могут выдавать восьмибитные значения («источники данных»), а также четыре устройства, которые могут принимать восьмибитные значения («приёмники данных»). Устройства могут быть любого рода — комбинационные, последовательностные, регистры, ОЗУ, устройства ввода и вывода, и т.д. Наша задача — иметь возможность по выбору передавать восьмибитные значения от любого устройства-источника к любому устройству-приёмнику. Совокупность устройств, необходимых для реализации такой возможности называется *шиной* (англ. bus, bidirectional universal switch — двунаправленный универсальный коммутатор). Кроме того, часто шиной называют непосредственно многобитный пучок проводов, несущий сигналы от источников к приёмникам. Судя по переводу английского названия шины, передача данных должна быть двунаправленной, то есть одно и то же устройство может быть как источником, так и приёмником данных. На первый взгляд, наш пример подразумевает только однонаправленную передачу, но это не так: устройство в нашем примере может выступать в качестве

источника и приёмника одновременно, если мы будем считать источником данных его выходы, а приёмником данных — его же входы. В цифровой технике одни и те же контакты могут быть входами и выходами одновременно (мы это увидим на примере порта данных ОЗУ в следующей лабораторной работе), однако, разбирая работу шины, мы будем жёстко разграничивать входные и выходные контакты — с одной стороны, это позволить сделать объяснение более ясным, а с другой стороны мы ограничены тем, что создаваемые пользователем контакты в Logisim не могут быть входами и выходами одновременно.

Рассмотрим способ реализации примитивной шины с использованием мультиплексора (рис. 3.19). Как правило, этот мультиплексор строится с использованием управляемых буферов — такая реализация оказывается гораздо более компактной и экономичной, так как мультиплексор, управляющий многобитными значениями, представляет собой группу из «однобитных» мультиплексоров, количество которых равно разрядности этого значения, и использование схемы с управляемыми буферами очень значительно снижает суммарное количество элементов.

Двухбитный вход «sourceCode» на схеме задаёт номер (код) устройства-источника, значение с выхода которого подаётся в данный момент на шину. Если максимальное количество устройств-источников равно n, а разрядность входа «sourceCode» равна m, то должно выполняться соотношение $2^m \geq n$. Очевидно, что в каждый момент времени происходит передача значения только от одного устройства.

На представленной схеме в любой момент времени все устройства-приёмники получают данные с шины. Если устройства-приёмники имеют тактовые входы (т. е. они синхронные), а это типично при использовании шины, то можно управлять тем, какие из устройств будут получать синхронизирующие импульсы. На схемах ниже представлены две реализации такой возможности: для случая, когда в каждый такт только один из приёмников может получать тактовый импульс (рис. 3.20), и для случая, когда тактовый импульс может быть передан любой комбинации устройств-приёмников (рис. 3.21).

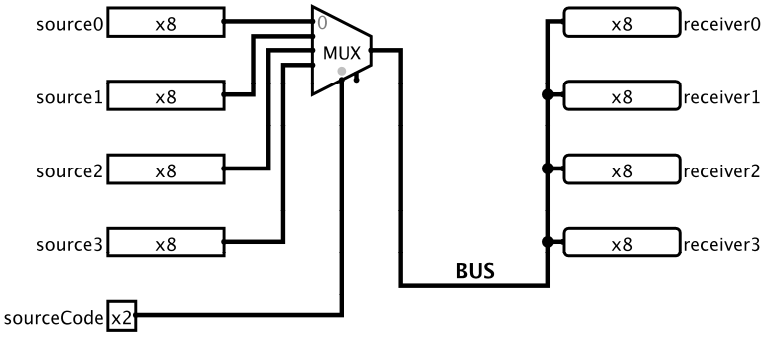



*Рис. 3.19.* Шина с использованием мультиплексора

В первом варианте вход «receiverCode» задаёт номер (код) устройства, которому посылается тактовый импульс; его разрядность определяется по тому же соотношению, что и для входа «sourceCode». Во втором варианте вход «receiversFlags» задаёт комбинацию устройств для получения тактового импульса: единица в каждом отдельном бите этого входа означает, что устройство с соответствующим номером получит тактовый импульс.

**Задание 1.** Спроектировать схему для решения следующей задачи. Необходимо иметь возможность передавать последовательность 8-разрядных значений по одному проводу (плюс провод для передачи синхронизирующих импульсов), то есть нужно производить мультиплексирование сигнала.

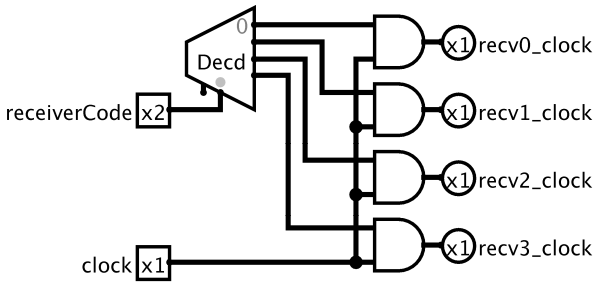



*Рис. 3.20.* Устройство управления тактовыми импульсами, вариант 1

Схема процесса мультиплексирования 4-разрядных значений показана в лабораторной работе 3 (рис. 3.10). Для решения задачи нужно разделить эту схему на две подсхемы — передатчик мультиплексированного сигнала и его приёмник, и изменить разрядность входного и выходного значений на 8. Кроме того, и приёмник и передатчик должны передавать во внешнюю схему импульс, сигнализирующий о том, что процесс передачи очередного 8-разрядного значения завершён, на вход передатчика можно подавать новое значение, а с выхода приёмника — считывать новое принятое значении. Этот импульс можно снимать с выхода «перенос» счётчиков в составе передатчика и приёмника. Новые значения пусть генерируются компонентом «Генератор случайных чисел» из библиотеки «Память», а с выхода приёмника — записываются в 8-битный регистр. На внешней схеме кроме передатчика и приёмника должны присутствовать генератор случайных чисел, тактовый генератор, регистр, в который записываются принятые значения, и главное — провод, по которому передаётся мультиплексированный сигнал (и провод, несущий синхронизирующие импульсы).

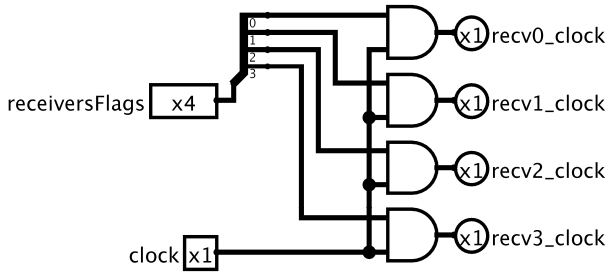



*Рис. 3.21.* Устройство управления тактовыми импульсами, вариант 2

**Задание 2.** Спроектировать устройство, поведение которого описано в таблице 3.8. Все устройства должны выполнять свою задачу потактово. Вариант определяется по последней цифре номера студента в списке группы. Все регистры имеют разрядность 32 бита.

Может быть полезным рассмотреть общий принцип решения каждой из этих задач всей группой, а выполнять реализацию — индивидуально.

*Таблица 3.8.* Варианты задания 2 для лабораторной работы 5

| Вариант | Поведение устройства |
|---|---|
| **1** | Вычисляет факториал числа, подаваемого на вход устройства, потактово умножая содержимое одного регистра на содержимое второго, и увеличивая значение второго на 1. |
| **2** | Выполняет деление с остатком значения одного регистра на значение второго, потактово вычитая делитель из делимого и сравнивая разность с нулём. |
| **3** | Вычисляет сумму первых n членов арифметической прогрессии с разностью d, потактово увеличивая значение одного регистра на значение второго, и увеличивая значение второго на d. n и d задаются входами устройства. |
| **4** | Вычисляет сумму первых n членов геометрической прогрессии со знаменателем q, потактово увеличивая значение одного регистра на значение второго, и увеличивая значение второго в q раз. n и q задаются входами устройства. |
| **5** | Имеется 16 констант с заранее введёнными 32-битными значениями. Устройство находит номер константы с наибольшим значением, реализуя классический алгоритм. |

*Окончание таблицы 3.8*

| | |
|---|---|
| **6** | Играет само с собой в «камень, ножницы, бумага»: каждый такт случайным образом генерируются два значения в диапазоне [0; 2], сравниваются по правилам игры, и на выход выдаётся код «победителя» (или код ничьи). |
| **7** | Находит количество единиц в двоичном представлении значения регистра, потактово перебирая биты значения. |
| **8** | Проверяет качество работы компонента «Генератор случайных чисел»: на протяжении большого числа тактов считает количество выданных им 32-битных значений больших и меньших среднего, и выводит разность между этими количествами. |
| **9** | Циклически заполняет пять регистров случайными значениями. Останавливается, когда все пять значений оказываются расположенными в порядке возрастания. |
| **10** | Девять D-триггеров расположены квадратом 3 на 3. По нечётным тактам устройство заполняет их случайными значениями, а по чётным — «поворачивает» значения восьми регистров вокруг центрального на 45 градусов по часовой стрелке. |

**Задание 3.** Спроектировать в виде отдельной схемы реализацию шины; в неё также должен быть включен первый вариант устройства управления тактовыми импульсами. Разрядность данных шины и количество устройств-источников и устройств-приёмников приведены в таблице 3.9. Для демонстрации работоспособности шины нужно на внешней схеме подключить к ней регистры с заранее занесёнными в них произвольными значениями, и каждый такт подавать на входы sourceCode и receiverCode случайные номера устройств. Попробуйте использовать некоторые регистры и как устройства-источники, и как устройства-приёмники одновременно. Проследите за перемещением значений по регистрам.

*Таблица 3.9.* Варианты задания 3 для лабораторной работы 5

| Вариант | Разрядность данных | Количество устройств-источников | Количество устройств-приёмников |
|---|---|---|---|
| **1** | 16 | 4 | 16 |
| **2** | 8 | 8 | 8 |
| **3** | 32 | 16 | 4 |
| **4** | 24 | 8 | 8 |
| **5** | 8 | 16 | 4 |
| **6** | 24 | 4 | 16 |
| **7** | 16 | 16 | 8 |
| **8** | 32 | 8 | 4 |
| **9** | 16 | 4 | 8 |
| **10** | 8 | 16 | 8 |

**Требования к выполнению работы:** все задания выполняются в одном файле проекта Logisim; каждое самостоятельное устройство должно быть оформлено в виде отдельной схемы с осмысленным названием всех входов и выходов и самой схемы. Для защиты каждого задания нужно продемонстрировать работоспособность каждого отдельного устройства на тестовой схеме.

## 3.6. Лабораторная работа 6. Сложные устройства памяти: ОЗУ и ПЗУ

*Время на выполнение: 2 часа*

Для хранения больших объёмов данных в сложных цифровых устройствах используют *ОЗУ* (оперативное запоминающее устройство, англ. RAM, random access memory) и *ПЗУ* (постоянное запоминающее устройство, англ. ROM, read-only memory). Главное отличие между ними — ОЗУ позволяет динамически (оперативно) изменять своё состояние, а при отключении питающего напряжения сохранённая в нём информация теряется, а ПЗУ сохраняет информацию и после отключения питания (т.н. «энергонезависимая память»), но эта информация статична (записывается один раз и не может быть оперативно перезаписана).

Физически ячейка памяти для хранения одного бита в составе ОЗУ может быть триггером на логических элементах (память статического типа, англ. SRAM), но чаще — элементарным устройством из одного транзистора и одного конденсатора (память динамического типа, англ. DRAM). Память динамического типа компактнее, дешевле, но медленнее из-за того, что время заряда или разряда конденсатора больше, чем задержки логических элементов в составе триггера.

Существует большое количество разных физических реализаций ПЗУ: масочные (ROM), программируемые (PROM), перепрограммируемые (EPROM), электрически стираемые перепрограммируемые (EEPROM), и некоторые другие. Флеш-память, в частности, является разновидностью EEPROM.

Логическая структура обоих видов ЗУ представляет собой линейную последовательность отдельных ячеек памяти фиксированной разрядности. Как правило, разрядность ячейки (этот параметр ЗУ называется «*разрядность данных*») является степенью двойки (8 битов, 16 битов, 32 бита, и т.д.), однако могут быть исключения. Длина последовательности ячеек определяется *разрядностью адреса* запоминающего устройства: если разрядность адреса равна n, то количество ячеек памяти в устройстве равно $2^n$.

Простейшее ОЗУ имеет многоразрядный вход адреса (A), значение на котором определяет номер ячейки, к которой мы обращаемся в данный момент; многоразрядный вход данных (D), на который подаётся значение для записи в выбранную ячейку; многоразрядный выход данных (D), на который выдаётся значение, сохранённое в текущей ячейке; и в случае

синхронного ОЗУ — тактовый вход (в Logisim обозначается треугольником), при срабатывании которого в выбранную ячейку записывается новое значение. Простейшее ПЗУ имеет только вход адреса и выход данных — поскольку запись в ПЗУ невозможна, вход данных и тактовый вход не нужны; значение на выходе данных обновляется сразу после поступления нового значения на вход адреса.

В Logisim также предусмотрен особый *интерфейс данных*, в случае использования которого ОЗУ имеет единый порт для записи и чтения данных (то есть он является одновременно входом и выходом). Для работы с таким интерфейсом данных нужно использовать компонент у*правляемый буфер*. Подробнее про этот интерфейс можно прочитать в статье «ОЗУ» справки по библиотеке Logisim (библиотека «Память»), а про управляемый буфер — в статье «Управляемый буфер/инвертор» (библиотека «Элементы»). В случае использования единого порта для чтения и записи, в Logisim можно выбрать между синхронным и асинхронным ОЗУ. Значительная часть промышленно выпускаемых модулей памяти — асинхронные, с единым портом.

Редактировать содержащиеся в ОЗУ или ПЗУ значения в Logisim можно щёлкнув на соответствующем компоненте правой кнопкой мыши и выбрав пункт контекстного меню «Редактировать содержимое...». После этого откроется встроенный в Logisim шестнадцатеричный редактор. Содержимое ПЗУ сохраняется в файле проекта Logisim (*.circ), и является атрибутом компонента ПЗУ (иными словами, однажды отредактировав содержимое ПЗУ, можно не беспокоиться за его сохранность). С ОЗУ ситуация иная — информация в ОЗУ динамически меняется, а при сбросе моделирования или перезагрузке проекта всё содержимое ОЗУ заполняется нулями. Однако в Logisim есть возможность сохранять и загружать содержимое ОЗУ и ПЗУ, используя отдельные файлы — файлы *образов памяти*. Это делается через пункты контекстного меню «Сохранить образ...» и «Загрузить образ...». Файлы образов Logisim — это обычные текстовые файлы, в которых данные записаны в шестнадцатеричном виде. Более подробно про загрузку и сохранение файлов образов и их формат можно прочитать в статье «Всплывающие меню и файлы» раздела «Компоненты памяти» руководства пользователя Logisim.

Кроме прочих, существует немного необычное использование ПЗУ. К нему можно прийти, если ответить на вопрос «является ли ПЗУ последовательностным логическим устройством?» С одной стороны, ПЗУ — это устройство памяти, а значит оно, скорее всего, последовательностное. Однако из определения последовательностного

устройства ясно, что его внутреннее состояние изменяется со временем. Для ПЗУ это не так — сохранённая в нём информация жёстко «зашита» и не может быть изменена, а значит, значения на выходах зависят только от значений на входах, то есть ПЗУ — комбинационное устройство. Отсюда вытекает вывод, что оно реализует набор булевых функций от значений на своих входах. Входами являются отдельные биты многоразрядного входа адреса, а выходами — отдельные биты многоразрядного выхода данных. Количество однобитных входов этого комбинационного устройства равно разрядности адреса ПЗУ, а количество выходов (т. е. булевых функций) равно разрядности данных ПЗУ. Зная это, можно использовать ПЗУ со введёнными в него данными вместо сложной схемы на логических элементах. На практике это почти всегда плохое решение — ПЗУ дороже, медленнее и потребляет больше энергии, чем комбинационное устройство, собранное на логических элементах. Но всё же в некоторых ситуациях это может быть оправданно. Например, в Logisim скорость моделирования сильнее зависит от количества элемента, чем от их сложности. Поэтому в ситуациях, когда скорость моделирования критична (например, при моделировании целого процессора), может быть разумно заменять комбинационные схемы из логических элементов на ПЗУ с введённой в него таблицей истинности.

**Задание 1.** Спроектировать устройство, заполняющее ОЗУ случайными значениями (с выхода компонента «Генератор случайных чисел» из библиотеки «Память»). Интерфейс данных ОЗУ (в этом задании и во всех последующих) — «Один асинхронный порт чтения/записи». Разрядность адреса ОЗУ — 10 битов, разрядность данных — 8 битов. После заполнения сохранить образ памяти в файл.

**Задание 2.** Спроектировать устройство, потактово считывающее содержимое одного ОЗУ, вычисляющее противоположное по знаку значение (в дополнительном коде) для каждого считанного значения, и записывающее результат во второе ОЗУ. Параметры ОЗУ — из задания 1. Перед началом считывания загрузить в ОЗУ-источник образ памяти, сохранённый в процессе выполнения задания 1; после копирования сохранить содержимое второго ОЗУ в отдельный файл.

**Задание 3.** Спроектировать устройство, потактово считывающее содержимое двух ОЗУ и складывающее считанные значения. Параметры ОЗУ — из задания 1. Перед считыванием загрузить в первое ОЗУ образ памяти, сохранённый в процессе выполнения задания 1, а во второе — сохранённый в процессе выполнения задания 2. Показать наглядно, что

результат сложения всегда равен нулю.

**Задание 4.** Спроектировать устройство, осуществляющее копирование содержимого одного ОЗУ в другое по одному проводу (плюс провод для передачи синхронизирующих импульсов), то есть нужно производить мультиплексирование сигнала. Параметры ОЗУ — из задания 1. Для реализации устройства нужно использовать схемы, созданные в результате выполнения задания 1 лабораторной работы 5. Перед копированием загрузить в первое ОЗУ образ памяти, сохранённый в процессе выполнения задания 1.

**Задание 5.** Спроектировать комбинационное устройство, описание которого приведено в задании 1 лабораторной работы 1, на основе ПЗУ (введя в ПЗУ таблицу истинности). Продемонстрировать работоспособность устройства. Один из способов выполнения этого задания — спроектировать небольшую схему, которая последовательно подаёт на входы устройства из лабораторной работы 1 и на адресный вход ОЗУ многобитные значения соответствующей разрядности, а многобитные значения с выходов этого устройства подаёт на вход данных ОЗУ для записи. После перебора всех вариантов входных значений содержимое ОЗУ можно сохранить в файл-образ, а затем загрузить этот образ в ПЗУ.

**Требования к выполнению работы:** все задания выполняются в одном файле проекта Logisim; каждое самостоятельное устройство должно быть оформлено в виде отдельной схемы с осмысленным названием всех входов и выходов и самой схемы. Для защиты каждого задания нужно продемонстрировать работоспособность каждого отдельного устройства на тестовой схеме.

## 3.7. Лабораторная работа 7. Устройства ввода и вывода

*Время на выполнение: 4 часа*

Для ввода и вывода данных (то есть для взаимодействия схемы с пользователем) в Logisim существует несколько компонентов, они представлены в библиотеке «Ввод/вывод». Довольно полную информацию о них можно получить в справке Logisim по этой библиотеке; для выполнения этой лабораторной работы рекомендуется прочитать этот раздел справки Logisim.

Самое примитивное устройство ввода в Logisim — компонент «*Кнопка*». Он имеет единственный однобитный выход, на котором «0» в нормальном состоянии, и «1», если кнопка нажата. Для нажатия на кнопку нужно воспользоваться инструментом «Нажатие» (как и всякий раз, когда нужно провзаимодействовать с компонентом в схеме).

Простейшее устройство вывода — компонент «*Светодиод*». Когда на его единственный однобитный вход поступает «1», светодиод «загорается», при подаче на вход «0» светодиод потушен. Цвет включенного и выключенного светодиода можно настраивать.

Классическое устройство отображения информации — компонент «*7-сегментный индикатор*», хорошо знакомый по бытовой цифровой технике. Символы (как правило, цифры) формируются из семи сегментов и десятичной точки, каждый из которых может быть подсвечен посредством подачи «1» на соответствующий однобитный вход. Соответствие между входами и сегментами, а также принятые буквенные обозначения сегментов показаны на рисунке 3.22. Такой индикатор способен формировать все шестнадцатеричные цифры, поэтому было бы удобно иметь комбинационное устройство, принимающее на входе четырёхбитное значение (шестнадцатеричную цифру), и имеющее семь выходов на сегменты индикатора. Его несложно спроектировать, начав с составления таблицы истинности (семь булевых функций от четырёх переменных). Далее можно реализовать это устройство на логических элементах, или введя таблицу истинности в ПЗУ.

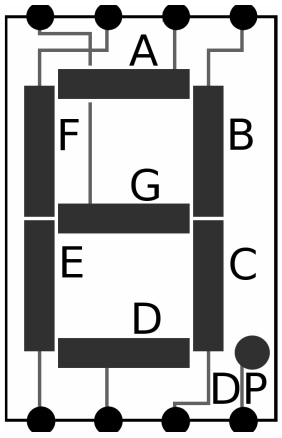


*Рис. 3.22.* Соответствие между входами и сегментами

Однако в библиотеке Logisim уже есть компонент «*Шестнадцатиразрядный индикатор*»,

принимающий на входе четырёхбитное значение, и отображающий соответствующее значение с помощью 7-сегментного индикатора. Расположив несколько шестнадцатеричных индикаторов вряд и подключив их к последовательным группам из четырёх битов многоразрядного значения с помощью разветвителей, можно отображать многоразрядные значения в шестнадцатеричном представлении.

Отображать многоразрядные значения в десятичном представлении немного сложнее. Пока четырёхразрядное значение не превышает число девять, его отображение с помощью шестнадцатеричного индикатора можно интерпретировать как десятичную цифру. Если производить целочисленное деление многоразрядного значения на степени числа десять, то частное такого деления — десятичная цифра соответствующего разряда десятичного представления числа, а остаток деления можно снова целочисленно делить на меньшую степень числа 10, и т.д. Пример реализации отображения десятичного представления восьмибитного числа с помощью шестнадцатеричных индикаторов показан на рисунке 3.23. Значения 64 и 0a на схеме — это числа 100 и 10 в шестнадцатеричном представлении. Заметим, что реализация на делителях — далеко не самое экономичное решение, так как делители — весьма сложные устройства.

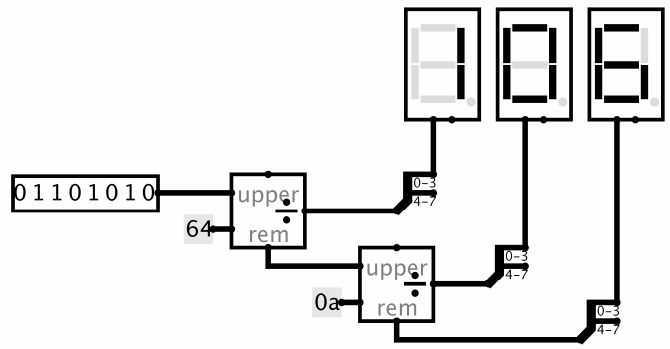



*Рис. 3.23.* Схема для отображения десятичного числа

Для отображения графической информации в Logisim существует компонент «*Светодиодная матрица*». Он представляет собой прямоугольную сетку точек с возможностью подсвечивать каждую в отдельности (единственным цветом). Максимальный размер сетки — 32x32 точки. Управление свечением точек может осуществляться одним из трёх способов в зависимости от значения атрибута «Формат входа»:

- «Столбцы»: входы расположены вдоль южного края компонента; один многобитный вход для каждого столбца матрицы.
- «Строки»: входы расположены вдоль западного края компонента; один многобитный вход для каждой строки матрицы.

- «Выбор Строки/Столбцы»: два входа на западном крае компонента. Верхний многобитный вход имеет столько битов, сколько столбцов в матрице; нижний многобитный вход имеет столько битов, сколько строк в матрице.

В первых двух форматах входа к матрице непосредственно подводится столько бит, сколько точек в матрице. Такой вариант крайне трудно осуществим на практике, и мы не будем использовать его в работе; как правило, промышленные матрицы имеют формат входа, соответствующий варианту «Выбор Строки/Столбцы» в Logisim. При таком варианте для формирования изображения используется *развёртка* (англ. *scanning*): в течение одного такта только один бит многобитного входа, определяющего подсвечиваемые строки, имеет значение «1». В этот момент на другой многобитный вход, определяющий подсвечиваемые столбцы, подаётся значение, в котором каждый бит «1» соответствует точке выбранной строки, которая должна быть подсвечена. После поочерёдного прохождения всех строк, снова подсвечивается первая, и т.д. Точки (пиксели) в составе реальных матриц не гаснут мгновенно, и за счёт этой инерции при прохождении развёртки не возникает мерцания. Чтобы сымитировать это, у компонента «Светодиодная матрица» в Logisim есть атрибут «Продолжительность свечения», который определяет, сколько тактов точка остаётся подсвеченной после подачи на неё единицы. Это довольно грубое приближение, однако оно позволяет успешно формировать развёртку. Чтобы полностью избежать мерцания, значение атрибута «Продолжительность свечения» должно быть равно удвоенному количеству строк в матрице. На рисунке 3.24 изображена примитивная схема, использующая четырёхбитный счётчик и декодер, формирующую развёртку, и поочерёдно подающая на вход матрицы значения из ПЗУ. Разрядность адреса ПЗУ (4 бита) позволяет адресовать 16 значений (по числу строк в матрице), а разрядность данных (8 бит) равна количеству столбцов в матрице. Значения в ПЗУ введены таким образом, чтобы при их последовательном выводе в строки матрицы формировалось нужное

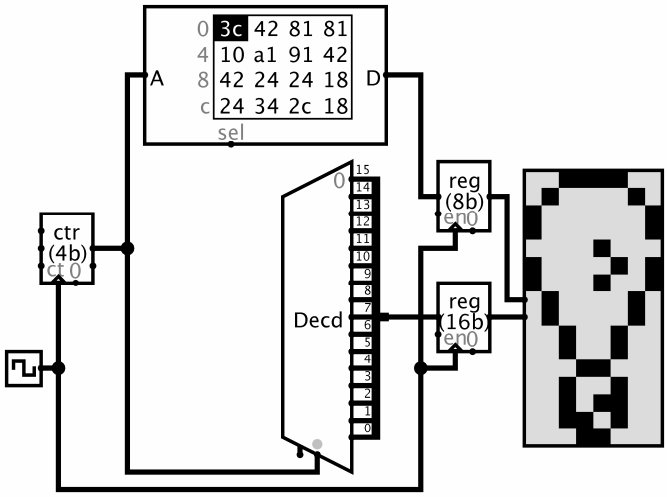



*Рис. 3.24.* Схема для формирования изображения на матрице

изображение. Перед обоими входами матрицы включены регистры соответствующей разрядности. Они позволяют подавать новый номер строки и значение, определяющее новое содержимое этой строки, строго одновременно (синхронно) — с поступлением на регистры нового тактового импульса. Если эти значения будут приходить с ПЗУ и декодера не одновременно, то изображение будет формироваться неправильно. Обратите внимание, что в этой схеме разветвитель не только объединяет 16 однобитных значений в пучок, но и меняет их порядок на противоположный, чтобы развёртка пробегала строки сверху-вниз, а не наоборот.

В Logisim есть два компонента для работы с текстовыми символами - «*Клавиатура*» и «*Терминал*». Компонент «Клавиатура» позволяет схеме считывать символы с клавиатуры, если только они представимы в 7-разрядном ASCII коде (в него входят буквы латинского алфавита, цифры и специальные символы; кириллица в него не входит). После нажатия на компонент с помощью инструмента «Нажатие», пользователь может печатать символы, которые накапливаются в буфере. ASCII значение крайнего левого символа буфера постоянно поступает на крайний правый выход. Когда срабатывает тактовый вход, крайний левый символ исчезает из буфера и новый левый символ поступает на крайний правый выход. Поддерживаемые символы — все печатаемые ASCII символы, а также пробел, символ перехода на новую строку (Enter), Backspace и Control-L (очистка экрана). В дополнение, клавиши «стрелка влево» и «стрелка вправо» перемещают курсор внутри буфера, а клавиша Delete удаляет символ справа от курсора (если он есть).

Компонент «Терминал» принимает последовательность ASCII кодов и при срабатывании тактового входа отображает каждый печатаемый символ, поступающий на его вход. Если текущая строка становится полной, то курсор перемещается на следующую строку, возможно прокручивая все текущие строки вверх, если курсор уже находился в нижнем ряду. Кроме того, компонент соответствующим образом реагирует на дополнительные коды, описанные выше.

На рисунке 3.25 приведена примитивная схема, которая каждый такт выводит на терминал очередной символ, введённый пользователем с клавиатуры.

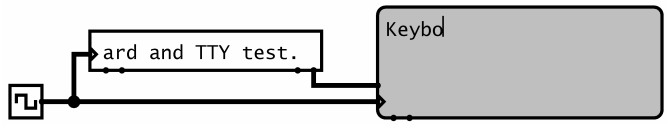



*Рис. 3.25.* Схема, выводящая введённые символы

**Задание 1.** Спроектировать в виде отдельной схемы комбинационное устройство, принимающее на входе 4-битное значение, и имеющее семь однобитных выходов. При подключении к выходам 7-сегментного индикатора, он должен отображать шестнадцатеричное представление значения на входе. Допустимо реализовать устройство на логических элементах или на основе ПЗУ. Отредактируйте в Logisim внешний вид подсхемы (в частности, расположение выходных контактов) таким образом, чтобы 7-сегментный индикатор мог подключаться к выходам подсхемы вообще без проводов, но при этом индикатор и подсхема не должны перекрываться.

**Задание 2.** Спроектировать комбинационное устройство, принимающее на входе беззнаковое 16-разрядное значение, и выводящее его десятичное представление на пять 7-сегментных индикаторов, каждый из которых подключен с помощью устройства из задания 1.

**Задание 3.** Спроектировать устройство для реализации следующей ситуации. Имеется два 16-разрядных регистра. Пользователь имеет в распоряжении четыре кнопки. Нажимая на первую или вторую, пользователь уменьшает или увеличивает значение первого регистра на значение второго регистра. Нажимая на третью или четвёртую кнопки, пользователь уменьшает или увеличивает значение второго регистра в 10 раз. Начальное значение второго регистра — 1, и оно не может опуститься ниже этого значения. Кнопки должны иметь осмысленные метки. Содержимое регистров отображается с помощью устройств из задания 2. Для арифметических действий допустимо использовать любые компоненты из библиотеки Logisim. Обратите внимание, что это устройство не требует наличия тактового генератора — тактовые импульсы для регистров генерируются нажатием кнопок.

**Задание 4.** Спроектировать устройство, выводящее на светодиодную матрицу размером 32x32 какое-либо осмысленное изображение, предварительно записанное в ПЗУ.

**Задание 5.** Спроектировать устройство, выводящее на светодиодную матрицу размером 32x32 «белый шум» — случайные значения с выхода компонента «Генератор случайных чисел» из библиотеки «Память».

**Задание 6.** Спроектировать устройство, реализующее совместный доступ устройств ввода и вывода к ОЗУ. Разрядность адреса ОЗУ — 5 битов, разрядность данных — 7 битов. По нечётным тактам в ячейку ОЗУ со случайным адресом (сгенерированным генератором случайных чисел) записывается ASCII код символа из буфера компонента «Клавиатура», причём если буфер пуст (на выходе компонента — значение «0»), то запись не происходит. По чётным тактам происходит

вывод содержимого ОЗУ на светодиодную матрицу размером 7x32 с помощью развёртки, то есть каждый чётный такт — следующая строка точек.

**Задание 7.** Спроектировать устройство, принимающее вводимые пользователем с клавиатуры символы, определённым образом обрабатывающее их, и выводящее на терминал. Алгоритм обработки указан в таблице 3.10. Вариант определяется по последней цифре номера студента в списке группы. Подразумевается, что символы, для которых таблицей не предусмотрена обработка, выводятся без изменений.

По сути, задание сводится к проектированию комбинационного устройства, которое нужно разместить между компонентами «Клавиатура» и «Терминал». Поскольку количество символов в 7-битном наборе ASCII довольно велико (128), заполнять таблицу истинности для каждого символа слишком трудоёмко. Вместо этого для определения принадлежности введённого символа к определённой группе (прописные и строчные буквы, цифры, специальные символы) следует использовать компонент «Компаратор» из библиотеки «Арифметика».

Для получения таблицы соответствия 7-битных ASCII кодов и символов набора можно спроектировать схему, посылающую последовательные 7-битные значения на вход компонента «Терминал». Для обратной задачи (выяснения кода определённого символа) можно спроектировать другую схему — подключить выход компонента «Клавиатура» к какому-либо устройству отображения (например, к компоненту «Датчик»), вводить интересующие символы и считывать их коды.

*Таблица 3.10.* Варианты заданий для лабораторной работы 7

| Вариант | Алгоритм обработки символа |
|---|---|
| **1** | Заменять все прописные буквы строчными |
| **2** | Заменять все строчные буквы прописными |
| **3** | Заменять каждую цифру большей на единицу |
| **4** | Не выводить никаких символов кроме букв и цифр |
| **5** | Заменять каждую прописную букву следующей в алфавите |
| **6** | Заменять каждую строчную букву следующей в алфавите |
| **7** | Заменять прописные гласные строчными |
| **8** | Заменять строчные гласные прописными |
| **9** | Заменять все согласные на тире |
| **10** | Выводить вместо символа младшую половину его шестнадцатеричного представления |

**Требования к выполнению работы:** все задания выполняются в одном файле проекта Logisim; каждое самостоятельное устройство должно быть оформлено в виде отдельной схемы с осмысленным названием всех входов и выходов и самой схемы. Для защиты каждого задания нужно продемонстрировать работоспособность каждого отдельного устройства на тестовой схеме.

## 3.8. Лабораторная работа 8. Ассемблер.

*Время на выполнение: 8 часов*

*Ассемблер* (от англ. assembler — сборщик) — компьютерная программа, преобразующая исходный текст программы, написанной на языке ассемблера, в программу на машинном языке (то есть в последовательность инструкций процессора). *Язык ассемблера* — язык программирования низкого уровня, мнемонические команды которого соответствуют инструкциям процессора. *Инструкция процессора* — элементарная операция, которую может выполнить процессор. Различные процессоры имеют различные *наборы инструкций*, поэтому язык ассемблера не является универсальным, он зависит от конкретного процессора.

В данной лабораторной работе мы будем иметь дело с процессором lilpM32. Для работы понадобятся схема процессора, спроектированная в Logisim, и программа-ассемблер (lilpM32asm), позволяющая редактировать программы на языке ассемблера lilpM32 и преобразовывать их в файлы образов памяти для последующей загрузки в компонент ОЗУ схемы процессора. Они доступны на странице http://lilpm32.sourceforge.net/ru/.

Процессор lilpM32 разрабатывался для образовательных целей. Его набор инструкций и внутреннее устройство представляют собой очень сильно упрощённый вариант реализации архитектуры MIPS I. *MIPS* (англ. Microprocessor without Interlocked Pipeline Stages — микропроцессор без заблокированных стадий конвейера) — семейство процессоров, разработанных компанией MIPS Technologies, и являющихся реализацией архитектуры RISC. *RISC* (англ. Reduced Instruction Set Computer — компьютер с сокращённым набором инструкций) — архитектура процессора, в которой быстродействие увеличивается за счёт упрощения инструкций. В настоящее время различные реализации MIPS используются в основном во встроенных системах (в маршрутизаторах, шлюзах, игровых консолях; например, в Sony PlayStation Portable).

lilpM32 — 32-битный процессор. Это означает, что длина инструкции, а также разрядность обрабатываемых им данных — 32 бита. Формально, длина адреса ОЗУ — тоже 32 бита, но компонент ОЗУ в Logisim может иметь максимальную разрядность адреса 24 бита, поэтому фактически

при доступе к памяти учитываются только младшие 24 бита адреса. В lilpM32 нельзя получить доступ к отдельному байту памяти (как это обычно принято) — только к 32-разрядному слову. Значение, поступающее на адресный вход ОЗУ, задаёт номер 32-разрядного слова (а не байта) для чтения или записи. Таким образом, в нашем распоряжении примерно 16 миллионов 4-байтовых ячеек (то есть 64 мегабайта памяти). Для инструкций и для данных используется общая память, то есть lilpM32 имеет архитектуру фон Неймана.

Для работы с целыми числами и адресами lilpM32 имеет 32 32-битных регистра. Они объединены в *регистровый файл*. Номера и имена регистров принято начинать со знака доллара («$»); для ассемблера это обязательное условие. Регистр с номером 0 (`$0`) всегда содержит значение «0» (запись в него не имеет смысла). В регистр `$31` сохраняется адрес возврата из подпрограммы (функции). Эти два условия гарантируются аппаратной реализацией процессора. Кроме того, все остальные регистры имеют свои имена и назначение, однако это только соглашение, и сам процессор никак не контролирует выполнение этого соглашения — это задача программиста. В таблице 3.11 представлены названия регистров и их назначение.

*Таблица 3.11*. Назначение и названия регистров

| Название | Номер | Назначение | Сохраняется вызываемой функцией? |
|---|---|---|---|
| `$zero` | `$0` | константа 0 | - |
| `$at` | `$1` | временный для ассемблера | - |
| `$v0-$v1` | `$2-$3` | значения, возвращаемые функциями | - |
| `$a0-$a3` | `$4-$7` | аргументы функций | - |
| `$t0-$t7` | `$8-$15` | временные | - |
| `$s0-$s7` | `$16-$23` | сохраняемые временные | да |
| `$t8-$t9` | `$24-$25` | временные | - |
| `$k0-$k1` | `$26-$27` | зарезервированы для ядра ОС | - |
| `$gp` | `$28` | глобальный указатель на данные | да |
| `$sp` | `$29` | указатель стека | да |
| `$fp` | `$30` | указатель окна | да |
| `$ra` | `$31` | адрес возврата из функции | - |

lilpM32 имеет два регистра для сохранения двух частей 64-битного результата умножения и целочисленного деления: регистры LO и HI. Для ввода и вывода данных у процессора предусмотрено по четыре 32-битных буфера (регистра) ввода/вывода. Для доступа к этим буферам процессор имеет по четыре 32-битных входа и выхода для данных, тактовых входа, тактовых выхода, сигнализирующих о чтении данных процессором, и тактовых выхода сигнализирующих о записи данных процессором. При записи нового значения (после выполнения соответствующей инструкции) в буфер, процессор подаёт тактовый импульс на соответствующий тактовый выход. Чтобы записать новое значение в буфер ввода, нужно подать это значение на соответствующий вход для данных, а на тактовый вход подать тактовый импульс. После этого можно обращаться к записанному значению посредством соответствующей инструкции. Каждый раз, когда процессор считывает значение из буфера, он подаёт тактовый импульс на соответствующий тактовый выход. Буферы ввода/вывода можно использовать для подачи в процессор символов с клавиатуры, или для вывода информации на терминал или на светодиодную матрицу (непосредственно или используя отдельную видеопамять). Заметьте, что наличие буферов ввода/вывода и инструкций для работы с ними — отличительная особенность lilpM32; это сделано для упрощения. Как правило, ввод и вывод MIPS процессоры осуществляют несколько другими способами.

Кроме того, у процессора предусмотрен тактовый выход, который позволяет подавать во внешнюю схему импульсы непосредственно с тактового генератора, расположенного внутри схемы процессора.

Все инструкции процессора lilpM32 имеют длину 32 бита. В зависимости от типа инструкции (существуют R, I и J инструкции), эти 32 бита разбиваются на поля, каждое из которых несёт определённую информацию — код операции (opcode), номера регистров ($s, $t, $d), величину сдвига (shamt), код функции (funct), непосредственное значение (imm) и адрес перехода (addr). Для инструкций R-типа значение кода операции всегда равно нулю, а инструкция определяется по коду функции. Для инструкций I-типа сама инструкция содержит непосредственное 16-битное целочисленное значение, которое может быть интерпретировано (и расширено до 32-битного) как беззнаковое, или как знаковое, записанное в дополнительном коде (в зависимости от конкретной инструкции). Форматы инструкций представлены в таблице 3.12.

*Таблица 3.12.* Форматы инструкций

| **Тип** | **Формат (биты)** | | | | | |
|---|---|---|---|---|---|---|
| **R** | opcode (6) | $s (5) | $t (5) | $d (5) | shamt (5) | funct (6) |
| **I** | opcode (6) | $s (5) | $t (5) | imm (16) | | |
| **J** | opcode (6) | addr (26) | | | | |

Далее приводится список всех доступных инструкций lilpM32 с подробными описаниями (таблица 3.13). В конце таблицы располагаются *псевдоинструкции* (они отмечены прочерком в столбце «Формат»). Псевдоинструкции не являются инструкциями процессора; при компиляции ассемблер заменяет их определённой комбинацией настоящих инструкций. Использование псевдоинструкций облегчает написание и чтение кода.

*Таблица 3.13.* Набор инструкций процессора lilpM32

| **Синтаксис** | **Название** | **Действие** | **Формат / opcode / funct** | | |
|---|---|---|---|---|---|
| `addu $d,$s,$t` | Add unoverflow (Сложить без переполнения) | `$d = $s + $t` | R | 0x00 | 0x21 |
| Складывает два регистра, игнорируя переполнение. | | | | | |
| `subu $d,$s,$t` | Subtract unoverflow (Вычесть без переполнения) | `$d = $s - $t` | R | 0x00 | 0x23 |
| Вычитает два регистра, игнорируя переполнение. | | | | | |
| `addiu $t,$s,imm` | Add immediate unoverflow (Сложить непосредственное без переполнения) | `$t = $s + imm` | I | 0x09 | - |
| Складывает регистр с константой, игнорируя переполнение. `imm` — знаковое. | | | | | |
| `mult $s,$t` | Multiply (Умножить) | `LO = (($s * $t) << 32) >> 32;` `HI = ($s * $t) >> 32;` | R | 0x00 | 0x18 |
| Перемножает два регистра и помещает 64-битный результат в два специальных 32-битных регистра — `LO` и `HI`. | | | | | |

*Продолжение таблицы 3.13*

| | | | | | |
|---|---|---|---|---|---|
| `divu $s, $t` | Divide unsigned<br>(Делить беззнаковые) | `LO = $s / $t;`<br>`HI = $s % $t;` | R | 0x00 | 0x1b |
| Выполняет целочисленное деление над двумя регистрами и помещает частное в регистр `LO`, а остаток — в регистр `HI`. | | | | | |
| `lw $t,imm($s)` | Load word<br>(Загрузить слово) | `$t = Memory[$s + imm]` | I | 0x23 | - |
| Загружает в регистр слово, находящееся в ОЗУ по адресу `$s + imm`. `imm` — знаковое. | | | | | |
| `sw $t,imm($s)` | Store word<br>(Сохранить слово) | `Memory[$s + imm] = $t` | I | 0x2b | - |
| Сохраняет слово из регистра в ОЗУ по адресу `$s + imm`. `imm` — знаковое. | | | | | |
| `lui $t,imm` | Load upper immediate<br>(Загрузить непосредственное вверх) | `$t = imm << 16` | I | 0x0f | - |
| Загружает непосредственный 16-битный операнд в старшие 16 бит регистра. `imm` — беззнаковое. | | | | | |
| `mfhi $d` | Move from `HI`<br>(Копировать из `HI`) | `$d = HI` | R | 0x00 | 0x10 |
| Копирует значение из регистра `HI` в указанный регистр. | | | | | |
| `mflo $d` | Move from `LO`<br>(Копировать из `LO`) | `$d = LO` | R | 0x00 | 0x12 |
| Копирует значение из регистра `LO` в указанный регистр. | | | | | |
| `lfb $t,imm($s)` | Load from buffer<br>(Загрузить из буфера) | `$t = inbuf[$s + imm]` | I | 0x2d | - |
| Загружает в регистр значение из буфера, номер которого (от 0 до 3) задаётся значением `$s + imm` (используются только 2 младших бита этого значения). `imm` — знаковое. Чтение буфера происходит на стадии конвейера «Доступ к памяти». | | | | | |
| `stb $t,imm($s)` | Store to buffer<br>(Сохранить в буфер) | `outbuf[$s + imm] = $t` | I | 0x2e | - |
| Записывает значение регистра в буфер, номер которого (от 0 до 3) задаётся значением `$s + imm` (используются только 2 младших бита этого значения). `imm` — знаковое. Запись в буфер происходит на стадии конвейера «Доступ к памяти». | | | | | |
| `and $d,$s,$t` | And (И) | `$d = $s & $t` | R | 0x00 | 0x24 |
| Выполняет побитовое И над двумя регистрами и записывает результат в третий | | | | | |
| `andi $t,$s,imm` | And immediate<br>(И с непосредственным) | `$t = $s & imm` | I | 0x0c | - |
| Выполняет побитовое И над регистром и константой и записывает результат в регистр. `imm` — беззнаковое. | | | | | |

*Продолжение таблицы 3.13*

| | | | | | |
|---|---|---|---|---|---|
| `or $d,$s,$t` | Or (ИЛИ) | `$d = $s \| $t` | R | 0x00 | 0x25 |
| Выполняет побитовое ИЛИ над двумя регистрами и записывает результат в третий. | | | | | |
| `ori $t,$s,imm` | Or immediate (ИЛИ с непосредственным) | `$t = $s \| imm` | I | 0x0d | - |
| Выполняет побитовое ИЛИ над регистром и константой и записывает результат в регистр. `imm` — беззнаковое. | | | | | |
| `xor $d,$s,$t` | Exclusive or (Исключающее ИЛИ) | `$d = $s ^ $t` | R | 0x00 | 0x26 |
| Выполняет побитовое Исключающее ИЛИ над двумя регистрами и записывает результат в третий. | | | | | |
| `nor $d,$s,$t` | Nor (ИЛИ-НЕ) | `$d = ! ($s \| $t)` | R | 0x00 | 0x27 |
| Выполняет побитовое ИЛИ-НЕ над двумя регистрами и записывает результат в третий. | | | | | |
| `slt $d,$s,$t` | Set on less than (Установить, если меньше) | `$d = ($s < $t)` | R | 0x00 | 0x2a |
| Сравнивает значения двух регистров как знаковые (в дополнительном коде) и записывает результат в третий («1» — если значение первого меньше, «0» — в противном случае). | | | | | |
| `slti $t,$s,imm` | Set on less than immediate (Установить, если меньше, чем непосредственное) | `$t = ($s < imm)` | I | 0x0a | - |
| Сравнивает значения регистра и константы как знаковые и записывает результат в третий («1» — если значение регистра меньше, «0» — в противном случае). | | | | | |
| `sll $t,$s,shamt` | Shift left logical (Левый логический сдвиг) | `$t = $s << shamt` | R | 0x00 | 0x00 |
| Сдвигает значение регистра на `shamt` битов влево и записывает результат в другой регистр. Фактически, умножает значение регистра на $2^{\texttt{shamt}}$. `shamt` — беззнаковое. | | | | | |
| `srl $t,$s,shamt` | Shift right logical (Правый логический сдвиг) | `$t = $s >> shamt` | R | 0x00 | 0x02 |
| Сдвигает значение регистра на `shamt` битов вправо, заполняя освободившееся место нулями, и записывает результат в другой регистр. Фактически, делит значение регистра на $2^{\texttt{shamt}}$, если оно рассматривается как беззнаковое, или положительное знаковое. `shamt` — беззнаковое. | | | | | |
| `sra $t,$s,shamt` | Shift right arithmetic (Правый арифметический сдвиг) | `$t = ($s >> shamt) + sign_extension` | R | 0x00 | 0x03 |
| Сдвигает значение регистра на `shamt` битов вправо, заполняя освободившееся место битами знака, и записывает результат в другой регистр. Фактически, делит значение регистра на $2^{\texttt{shamt}}$, если оно рассматривается как знаковое. `shamt` — беззнаковое. | | | | | |

*Продолжение таблицы 3.13*

| | | | | | |
|---|---|---|---|---|---|
| `beq $s,$t,imm` | Branch on equal (Ветвление при равенстве) | `if ($s == $t) PC = PC + 1 + imm` | I | 0x04 | - |
| Переходит на выполнение инструкции по адресу `PC + 1 + imm` (где `PC` — текущее значение программного счётчика), если два регистра равны. `imm` — знаковое целое или метка. | | | | | |
| `bne $s,$t,imm` | Branch on not equal (Ветвление при неравенстве) | `if ($s != $t) PC = PC + 1 + imm` | I | 0x05 | - |
| Переходит на выполнение инструкции по адресу `PC + 1 + imm` (где `PC` — текущее значение программного счётчика), если два регистра не равны. `imm` — знаковое целое или метка. | | | | | |
| `j addr` | Jump (Прыжок) | `PC = addr` | J | 0x02 | - |
| Переходит на выполнение инструкции по адресу `addr`. `addr` — беззнаковое 26-битное значение или метка; поскольку разрядность адреса ОЗУ — 24 бита, используются только 24 младших бита `addr`. | | | | | |
| `jr $s` | Jump register (Прыжок к значению регистра) | `PC = $s` | R | 0x00 | 0x08 |
| Переходит на выполнение инструкции по адресу, хранящемуся в регистре. Используются только 24 младших бита этого значения. | | | | | |
| `jal addr` | Jump and link (Прыжок и связывание) | `$31 = PC + 4; PC = addr;` | J | 0x03 | - |
| Переходит на выполнение инструкции по адресу `addr` и записывает в регистр `$31` (`$ra`) адрес возврата. Используется для вызова подпрограмм. Возврат из подпрограммы осуществляется вызовом инструкции `jr $ra`.<br>`addr` — беззнаковое целое или метка, используются только 24 его младших бита. | | | | | |
| `nop` | No operation (Нет операции) | - | - | - | - |
| Не выполняет никакой операции. Часто используется в качестве «заглушки».<br>Заменяется ассемблером на:<br>`sll $0,$0,0` | | | | | |
| `move $t,$s` | Move (Копировать) | `$t = $s` | - | - | - |
| Копирует содержимое одного регистра в другой.<br>Заменяется ассемблером на:<br>`addiu $t,$s,0` | | | | | |
| `la $at,label` | Load address (Загрузить адрес) | `$at = label` | - | - | - |
| Загружает в регистр фактический адрес метки.<br>Заменяется ассемблером на:<br>`lui $at,label[31:16]; nop; nop; nop; ori $at,$at,label[15:0]` | | | | | |

*Продолжение таблицы 3.13*

| | | | | | |
|---|---|---|---|---|---|
| `li $at,imm` | Load immediate (Загрузить непосредственное) | `$at = imm` | - | - | - |
| Загружает в регистр непосредственное 32-битное знаковое значение.<br>Заменяется ассемблером на:<br>`lui $at,imm[31:16]; nop; nop; nop; ori $at,$at,imm[15:0]` | | | | | |
| `lli $t,imm` | Load lower immediate (Ветвление, если больше) | `$t = imm` | - | - | - |
| Загружает в регистр непосредственное 16-битное знаковое значение.<br>Заменяется ассемблером на:<br>`addiu $t,$0,imm` | | | | | |
| `bgt $s,$t,label` | Branch if greater than (Ветвление, если больше) | `bgt $s,$t,label` | - | - | - |
| Переходит на выполнение инструкции по адресу `label`, если значение одного регистра больше, чем значение другого.<br>Заменяется ассемблером на:<br>`slt $at,$t,$s; nop; nop; nop; bne $at,$zero,label` | | | | | |
| `blt $s,$t,label` | Branch if less than (Ветвление, если меньше) | `blt $s,$t,label` | - | - | - |
| Переходит на выполнение инструкции по адресу `label`, если значение одного регистра меньше, чем значение другого.<br>Заменяется ассемблером на:<br>`slt $at,$s,$t; nop; nop; nop; bne $at,$zero,label` | | | | | |
| `bge $s,$t,label` | Branch if greater than or equal (Ветвление, если больше или равно) | `bge $s,$t,label` | - | - | - |
| Переходит на выполнение инструкции по адресу `label`, если значение одного регистра больше или равно значению другого.<br>Заменяется ассемблером на:<br>`slt $at,$s,$t; nop; nop; nop; beq $at,$zero,label` | | | | | |
| `ble $s,$t,label` | Branch if less than or equal (Ветвление, если меньше или равно) | `if ($s <= $t)`<br>`PC = label` | - | - | - |
| Переходит на выполнение инструкции по адресу `label`, если значение одного регистра меньше или равно значению другого.<br>Заменяется ассемблером на:<br>`slt $at,$t,$s; nop; nop; nop; beq $at,$zero,label` | | | | | |

*Окончание таблицы 3.13*

| | | | | | |
|---|---|---|---|---|---|
| `mul $d,$s,$t` | Multiply and return low (Умножить и вернуть младшее) | `$d = (($s * $t) << 32) >> 32` | - | - | - |
| Перемножает значения двух регистров и записывает младшие 32 бита результата в третий. Заменяется ассемблером на: `mult $s,$t; nop; nop; nop; mflo $d` | | | | | |
| `div $d,$s,$t` | Divide and return quotient (Делить и вернуть частное) | `$d = $s / $t` | - | - | - |
| Выполняет целочисленное деление над двумя регистрами и помещает частное в регистр. Заменяется ассемблером на: `divu $s,$t; nop; nop; nop; mflo $d` | | | | | |
| `rem $d,$s,$t` | Divide and return reminder (Делить и вернуть остаток) | `$d = $s % $t` | - | - | - |
| Выполняет целочисленное деление над двумя регистрами и помещает остаток в регистр. Заменяется ассемблером на: `divu $s,$t; nop; nop; nop; mfhi $d` | | | | | |

Структурная схема внутреннего устройства процессора lilpM32 показана на рисунке 3.26. Эта схема показывает лишь самые ключевые моменты внутреннего устройства процессора; многие составные части процессора показаны условно, а некоторые — совсем опущены. Вы можете изучить внутреннее устройство процессора во всех подробностях, анализируя его схему в Logisim.

Процессор lilpM32, как и все MIPS (и вообще RISC) процессоры, использует конвейер. *Конвейер* (англ. pipeline) — это способ организации вычислений внутри процессора, при котором обработка инструкции разделяется на последовательность независимых стадий с сохранением результатов в конце каждой стадии. В lilpM32 таких стадий пять:

- получение инструкции (Instruction Fetch, IF)
- декодирование инструкции и чтение регистров (Instruction Decode & Register Fetch, ID)
- выполнение и вычисление адреса (Execute & Address Calc., EX)
- доступ к памяти (Memory Access, MEM)
- запись в регистр (Write Back, WB)

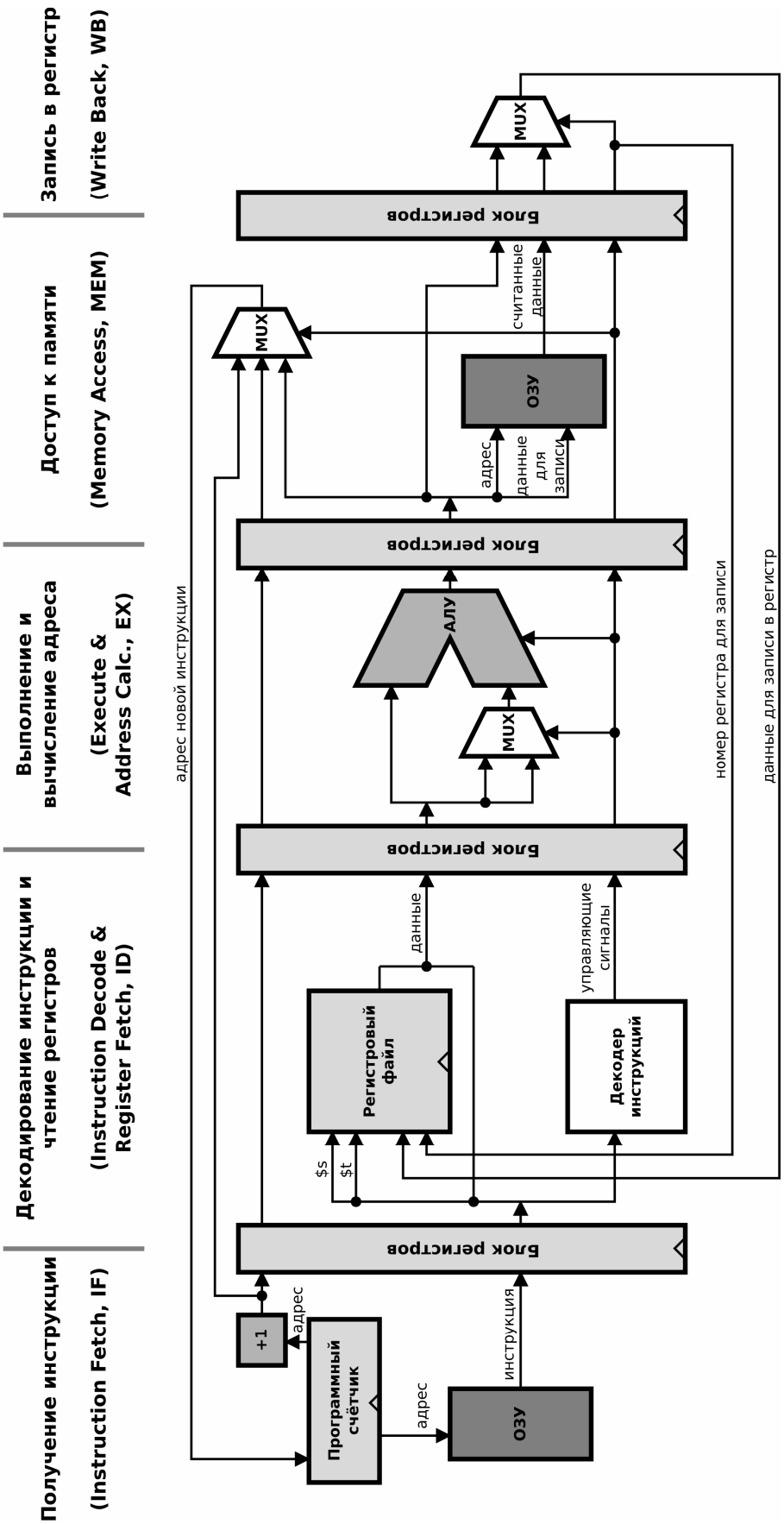



*Рис. 3.26.* Структурная схема процессора lilpM32

Каждая инструкция, считанная из ОЗУ, проходит по очереди все пять стадий, то есть на выполнение одной инструкции тратятся пять тактов процессора. Однако в каждый момент времени выполняется пять инструкций (по одной на каждой стадии). Таким образом, скорость выполнения инструкций — одна инструкция за такт. Блоки регистров между стадиями нужны чтобы сохранять промежуточные результаты выполнения каждой стадии: управляющие сигналы, считанные из устройств памяти данные, результат выполнения операции АЛУ, и т.д. Устройства, помеченные на схеме треугольником, имеют тактовый (синхронизирующий) вход; ОЗУ — асинхронное. ОЗУ изображено на схеме два раза только для наглядности; на самом деле это одно и то же устройство. За такт происходит два обращения к ОЗУ — считывается инструкция, расположенная по адресу, сохранённому в программном счётчике (на стадии получения инструкции), и считывается или записывается значение (на стадии доступа к памяти). В lilpM32 это достигается за счёт асинхронности ОЗУ — обращение к памяти производится и при переднем и при заднем фронте (этот принцип используется в памяти типа DDR — Double Data Rate).

Использование конвейера даёт большой прирост производительности — за один такт выполняется одна инструкция, и суммарная задержка устройств внутри одной стадии меньше, чем без использования конвейера (а значит тактовая частота — выше). Однако часто возникают ситуации, когда использование конвейера приводит к возникновению *конфликтов* (англ. hazard). Чтобы понять суть этого явления, рассмотрим таблицу, демонстрирующую состояние конвейера во времени (таблица 3.14).

*Таблица 3.14.* Обычное состояние конвейера

| **Номер инструкции** | **Состояние конвейера** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | IF | ID | EX | MEM | WB | | | | |
| **2** | | IF | ID | EX | MEM | WB | | | |
| **3** | | | IF | ID | EX | MEM | WB | | |
| **4** | | | | IF | ID | EX | MEM | WB | |
| **5** | | | | | IF | ID | EX | MEM | WB |
| **Номер такта** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |

Во время пятого такта (он выделен) весь конвейер заполнен инструкциями — первая инструкция проходит стадию WB (последнюю), а пятая — стадию IF (первую). Если продолжить добавлять инструкции (шестую, седьмую, и т.д.), то и в последующие такты конвейер будет полностью заполнен.

Конфликт может случиться, если мы попробуем выполнить вот такую последовательность инструкций (все символы строки, начиная с «#», являются комментариями):

```
addu $t1, $t2, $t3   # $t1 = $t2 + $t3
subu $t4, $t1, $t5   # $t4 = $t1 - $t5
```

Записывая такую последовательность, мы подразумеваем, что на момент выполнения второй инструкции первая уже будет выполнена, то есть регистр `$t1` будет содержать результат сложения. Что же произойдёт на самом деле? Запись результата сложения в регистр происходит на стадии WB, а чтение значения регистра (в нашем случае — `$t1`) — на стадии ID. Если внимательно посмотреть на таблицу состояний конвейера (таблица 3.15), то видно, что первая инструкция пройдёт стадию WB позже, чем вторая — стадию ID.

*Таблица 3.15.* Конфликт при чтении из регистра

| **Инструкция** | **Состояние конвейера** | | | | | |
|---|---|---|---|---|---|---|
| **addu** | IF | ID | EX | MEM | WB | |
| **subu** | | IF | ID | EX | MEM | WB |
| **Номер такта** | **1** | **2** | **3** | **4** | **5** | **6** |

Самый простой способ избежать конфликта — вставить между инструкциями три псевдоинструкции nop («заглушки»). В таком случае инструкции пройдут стадии WB и ID в нужном нам порядке (таблица 3.16), но процессор три такта будет простаивать (выполнять «отсутствие операции»). Самый правильный с точки зрения производительности способ обходить конфликты — вставлять между потенциально конфликтующими инструкциями другие инструкции программы, то есть изменять порядок следования инструкций определённым образом. Ассемблер lilpM32asm автоматически обнаруживает потенциально конфликтные ситуации такого рода (когда чтение регистра может произойти до необходимой записи в него) и вставляет столько инструкций nop, сколько необходимо (их число может быть меньше трёх, если программист вставил между ними другие инструкции).

*Таблица 3.16.* Обход конфликта при чтении из регистра

| **Инструк-ция** | **Состояние конвейера** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **addu** | IF | ID | EX | MEM | WB | | | | |
| nop | | IF | ID | EX | MEM | WB | | | |
| nop | | | IF | ID | EX | MEM | WB | | |
| nop | | | | IF | ID | EX | MEM | WB | |
| **subu** | | | | | IF | ID | EX | MEM | WB |
| **Номер такта** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |

Существует ещё один класс ситуаций, которые могут вызвать конфликт конвейера. Некоторые инструкции (`beq`, `bne`, `j`, `jr`, `jal`, `bgt`, `blt`, `bge`, `ble`) меняют значение программного счётчика. Записывая следующие за ними инструкции, мы предполагаем, что переход на новый адрес (то есть изменение значения программного счётчика) уже произошёл. Однако фактическое изменение значения происходит только на стадии MEM, а значит, ещё три идущие подряд инструкции будут получены для выполнения (это происходит на стадии IF) из «старого» места в ОЗУ. Ассемблер lilpM32asm принудительно вставляет три инструкции `nop` после инструкций, меняющих значение программного счётчика (таблица 3.17; подразумевается, что инструкция `addu` записана по адресу, на который произвела переход инструкция `j`).

*Таблица 3.17.* Обход конфликта при переходе на новый адрес

| **Инструк-ция** | **Состояние конвейера** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **`j`** | IF | ID | EX | MEM | WB | | | | |
| `nop` | | IF | ID | EX | MEM | WB | | | |
| `nop` | | | IF | ID | EX | MEM | WB | | |
| `nop` | | | | IF | ID | EX | MEM | WB | |
| **`addu`** | | | | | IF | ID | EX | MEM | WB |
| **Номер такта** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |

Почти все псевдоинструкции заменяются ассемблером на последовательность инструкций, содержащую три инструкции `nop` для предотвращения конфликтов. Если вы видите возможность заменить их инструкциями вашей программы, то с точки зрения производительности лучше не использовать псевдоинструкцию, а включить в программу её реализацию с использованием реальных инструкций с заменой инструкций `nop` на другие инструкции.

Образ памяти, полученный в результате работы ассемблера lilpM32asm будет содержать очень много инструкций `nop` — возможно, больше, чем полезных инструкций. Это чудовищным образом снижает производительность процессора. Такой примитивный способ обхода

конфликтов выбран в lilpM32asm в целях упрощения. Реальные MIPS процессоры имеют различные аппаратные механизмы для предотвращения конфликтов без простаивания процессора.

Рассмотрим подробнее синтаксис языка ассемблера lilpM32asm. Программа, написанная на этом языке ассемблера, может состоять из следующих элементов: инструкций, псевдоинструкций, *объявлений меток*, *директив* и комментариев.

Инструкции и псевдоинструкции были подробно рассмотрены выше. На каждой строке может быть не больше одной инструкции. Регистры можно указывать по номерам (`$0-$31`) или по именам (`$zero`, `$at`, `$t0-$t9`, `$ra`, и т.д.). Непосредственные значения (`imm` и `shamt`) можно задавать как в шестнадцатеричном виде (с префиксом «`0x`»), так и в десятичном. Если такое значение выходит за допустимый диапазон (например, указано отрицательное число для инструкций, ожидающих положительное), то ассемблер выдаст сообщение об ошибке. Параметры `imm` и `addr` для инструкций `beq`, `bne`, `j` и `jal` могут быть численными значениями или метками, причём если для инструкций `beq` и `bne` адрес перехода задан меткой, то переход осуществляется непосредственно на эту метку, то есть дополнительная единица к значению программного счётчика не прибавляется. Параметр `label` для псевдоинструкций `la`, `bgt`, `blt`, `bge` и `ble` может быть только меткой.

Строка кода может начинаться с *объявления метки*. Оно состоит из идентификатора метки (который может состоять из латинских букв, цифр, и символов подчёркивания, и не может начинаться с цифры), за которым следует двоеточие. При использовании в инструкциях, метка заменяется ассемблером на фактический адрес в памяти инструкции или области данных, следующей за объявлением метки.

Существуют две директивы, позволяющие располагать в памяти данные: «`.word`» и «`.ascii`». За директивой «`.word`» может следовать одно 32-битное знаковое целое число (в десятичном или шестнадцатеричном представлении). Это значение будет записано ассемблером в файл образа памяти. За директивой «`.ascii`» должна следовать взятая в кавычки строка, состоящая из символов 7-битного набора ASCII. Для таких строк поддерживаются управляющие последовательности, стандартные для языка C: «`\n`» (разделитель строк), «`\t`» (табуляция), «`\\`» (обратная косая черта), «`\"`» (кавычка), «`\'`» (апостроф), «`\0`» (терминальный ноль), и некоторые другие. Каждый символ строки будет расширен до 32-битного значения, и последовательность этих значений будет записана ассемблером в файл

образа памяти. Чтобы иметь возможность обращаться из инструкций к областям данных, созданным с помощью директив, перед каждой такой областью нужно ставить объявление метки. Области данных и инструкции могут чередоваться в программе в любом порядке, однако правильно располагать данные после всех инструкций — в противном случае процессор может начать выполнение данных, что приведёт к непредсказуемым последствиям.

Все символы строки, начиная с символа «#», считаются комментариями и игнорируются ассемблером.

Для демонстрации синтаксиса языка ассемблера и организации циклов рассмотрим пример написания программы. Подразумевается, что к младшим семи битам выхода и тактовому выходу, сигнализирующему о записи процессором нового значения, буфера с номером 0 подключен компонент «Терминал», то есть запись значений в этот буфер будет выводить на терминал соответствующий младшим семи битам этого значения символ. Задача — написать программу, которая будет потактово выводить на терминал тестовую строку. Её код будет выглядеть следующим образом.

```
la     $t0, test_text    # $t0 = address(test_text)
loop:
 lw    $t1, 0($t0)       # $t1 = Memory[$t0]
 addiu $t0, $t0, 1       # $t0++
 stb   $t1, 0($zero)     # buf[0] = $t1
 bne   $t1, $zero, loop  # if ($t1 != '\0') goto loop
end:
j      end
test_text:
.ascii "Testing text.\nThe next string.\0"
```

В конце программы имеется область данных с меткой `test_text`, содержащая тестовую строку. В начале программы записываем адрес этой области в регистр `$t0`, который в течение программы будет содержать адрес символа, выводимого на терминал. Начало цикла отмечено меткой `loop`. В начале цикла в регистр `$t1` считывается из ОЗУ очередное значение, содержащее символ, а значение регистра `$t0` увеличивается на единицу. Далее вызывается инструкция `bne`, которая заставит программу перейти к началу цикла, если значение символа не равно нулю (то есть если не достигнут последний символ — терминальный ноль. Обратите внимание, что в этой программе есть

несколько мест, где могут произойти конфликты при чтении значения регистра. Ассемблер автоматически устранит конфликты, добавив нужное количество заглушек между этими парами инструкций.

Кроме того, ассемблер будет вынужден поставить три заглушки сразу после инструкции `bne`.

**Задание 1.** Подключить к буферу ввода/вывода процессора компонент «терминал» так, чтобы младшие семь битов буфера интерпретировались как ASCII символы и выводились на терминал при каждой записи в буфер. Написать на языке ассемблера подпрограмму, которая принимает в качестве аргумента 32-битное беззнаковое целое число и выводит его десятичное представление на новую строку терминала. Продемонстрировать работоспособность подпрограммы, вызвав её несколько раз с различными аргументами. Вызов подпрограммы осуществляется с помощью инструкции `jal`, возврат из подпрограммы — с помощью инструкции `jr $ra`, а аргументы передаются через регистры `$a0-$a3`.

**Задание 2.** Подключить к младшим восьми битам буфера ввода/вывода компонент «Генератор случайных чисел», выдающий случайные 8-битные значения при каждом чтении буфера. Написать на языке ассемблера программу, которая считывает 60 таких случайных значений, определённым образом обрабатывает их, а затем эти значения и последовательность значений, полученных после обработки, выводит на терминал, вызывая подпрограмму из задания 1. Алгоритм обработки приведён в таблице 3.18. Вариант определяется по последней цифре номера студента в списке группы.

*Таблица 3.18.* Варианты заданий для лабораторной работы 8

| **Вариант** | **Алгоритм обработки** |
|---|---|
| **1** | Значения разбиваются на последовательные пары. Результат обработки — последовательность произведений чисел в парах, отсортированная по возрастанию. |
| **2** | Результат обработки — последовательность нечётных значений, отсортированная по возрастанию, и следующая за ней последовательность чётных значений, отсортированная по возрастанию. |

*Окончание таблицы 3.18*

| | |
|---|---|
| **3** | Значения разбиваются на последовательные тройки, и в каждой тройке определяется максимальное. Результат обработки — последовательность максимальных значений из троек, отсортированная по убыванию. |
| **4** | Значения разбиваются на последовательные пары, и для каждой пары находится остаток от деления первого числа на второе. Результат обработки — последовательность этих остатков, отсортированная по возрастанию. |
| **5** | Результат обработки — последовательность значений, десятичное представление которых содержит две цифры, отсортированная по убыванию. |
| **6** | Значения разбиваются на последовательные пары. Результат обработки — последовательность произведений чисел в парах, отсортированная по убыванию. |
| **7** | Значения разбиваются на последовательные пары, и для каждой пары находится остаток от деления первого числа на второе. Результат обработки — последовательность этих остатков, отсортированная по убыванию. |
| **8** | Результат обработки — последовательность чётных значений, отсортированная по убыванию, и следующая за ней последовательность нечётных значений, отсортированная по убыванию. |
| **9** | Значения разбиваются на последовательные тройки, и в каждой тройке определяется максимальное. Результат обработки — последовательность максимальных значений из троек, отсортированная по возрастанию. |
| **10** | Результат обработки — последовательность значений, десятичное представление которых содержит три цифры, отсортированная по возрастанию. |

**Задание 3.** Подключить к двум буферам ввода/вывода процессора компоненты «Клавиатура», и «Терминал». Написать на языке ассемблера программу, выполняющую те же преобразования над вводимыми с клавиатуры символами, что и в задании 7 лабораторной работы 7 (с учётом вариантов). Результат обработки должен выводиться на терминал.

**Задание 4.** К двум буферам ввода/вывода подключить ещё один компонент ОЗУ, который будет выполнять функции видеопамяти. Один буфер хранит адрес для записи в видеопамять, а другой — записываемое по этому адресу значение. К видеопамяти подключена светодиодная матрица, изображение на которую выводится из видеопамяти с помощью развёртки. Буферы и развёртка имеют раздельный доступ к видеопамяти — по нечётным тактам осуществляется запись из буферов, а по чётным — считывание значения развёрткой. Задача — написать на языке ассемблера программу, управляющую записью значений в буферы вывода таким образом, чтобы на светодиодной матрице поочерёдно формировались два изображения — нормальное и негативное. Параметры матрицы и изображение для вывода нужно взять те же, что и в задании 4 лабораторной работы 7.

**Требования к выполнению работы:** все задания выполняются в одном файле проекта Logisim; каждому заданию соответствует отдельная схема, в которой процессор lilpM32 (схема которого загружена как библиотека Logisim) присутствует в качестве подсхемы. Для защиты работы нужно предоставить коды всех программ на языке ассемблера и продемонстрировать их работоспособность в Logisim.

# 4. Курсовой проект

## 4.1. Общее описание проекта и требования

Основная задача курсового проекта — спроектировать в Logisim относительно сложное законченное цифровое устройство, выполняющее определённые функции, и возможно, применимое на практике. Это может быть электронная составляющая бытового устройства, система управления каким-либо процессом, игра, и т.д. Далее приводится список примеров устройств, реализацию которых возможно осуществить в рамках этого курсового проекта. Кроме назначения устройства в списке приводятся дополнительные требования к устройству, возможные элементы управления, некоторые идеи по реализации, и т.п. Более точное техническое задание должно быть подробно оговорено с преподавателем: вы можете внести какие-либо изменения в идеи по реализации устройства, приведённые в этом списке. Собственные предложения (т.е. устройства, не приведённые в этом списке) особо приветствуются, но и они, разумеется, должны быть подробно оговорены с преподавателем.

Кроме файла проекта Logisim с реализацией устройства, для защиты курсовой работы нужно выполнить отчёт по выполнению курсового проекта. В отчёте должен быть подробно изложен весь процесс проектирования устройства и описание логики его работы; приведены чертежи схем, таблицы истинности для комбинационных устройств, общие блок-схемы устройства, и т.д. При составлении отчета можно руководствоваться таким критерием: человек, знакомый с данным курсом, после прочтения вашего отчёта должен полностью понять все стадии проектирования устройства, логику его работы и назначение подсхем; и должен быть способен при желании повторить все шаги разработки самостоятельно.

Весь курсовой проект должен быть выполнен в виде одного файла проекта Logisim. Каждое самостоятельное устройство должно быть оформлено в виде отдельной схемы с осмысленным названием всех входов и выходов, устройств управления, и самой схемы. Функционирование (а значит и демонстрация возможностей) многих законченных устройств невозможно без внешней среды. Например, для устройства управления каким-либо процессом нужно не только спроектировать это устройство, но и воссоздать (смоделировать)

процесс, которым оно управляет (по крайней мере, ту часть процесса, с которой устройство непосредственно взаимодействует). В таком случае одна из схем проекта должна играть роль этого процесса (среды), а спроектированное устройство управления должно быть другой отдельной схемой, подающей и принимающей из среды некоторую информацию. Обе схемы, как правило, размещаются в качестве подсхем на главной схеме проекта; на ней же размещаются компоненты для взаимодействия с пользователем.

Обратите внимание, что любое взаимодействие с пользователем может осуществляться только с помощью компонентов из библиотеки «Ввод/вывод». Инструмент «Нажатие», компоненты «Датчик», «Контакт», и прочие подобные способы взаимодействия со схемой являются лишь удобствами, предоставляемыми Logisim, на практике подобных инструментов нет.

Чтобы графически изобразить какие-либо составляющие устройства или процесса, имеющие характерный внешний вид, рекомендуется воспользоваться встроенным в Logisim редактором внешнего вида подсхем.

## 4.2. Варианты устройств для реализации

### 4.2.1. Система управления светофорами

Нужно смоделировать систему управления светофорами на загруженном перекрёстке. Система должна иметь несколько режимов работы: «час пик», «середина дня», «ночь». На подходе к некоторым (редко используемым) пешеходным переходам имеется кнопка, которой пешеходы сигнализируют о своём присутствии. На подъезде к перекрёстку установлены магнитные датчики присутствия автомобиля. Анализируя состояние вышеописанных устройств ввода, система должна регулировать цикл светофоров таким образом, чтобы сократить простои всех участников дорожного движения. Каждый светофор должен быть снабжён индикатором, показывающим количество секунд до переключения сигнала светофора.

При проектировании такой системы нужно уделить особое внимание графическому оформлению перекрёстка в Logisim. Наличие и состояние как автомобилей, так и пешеходов должно наглядно отображаться.

### 4.2.2. Банкомат

Нужно спроектировать устройство для управления банкоматом. Устройства ввода — считыватель информации с карты и клавиши

управления. Устройства вывода — механизм выдачи купюр и какое-либо устройство отображении информации для пользователя. Кроме того, в виде отдельной подсхемы должен быть смоделирован процесс обмена данными с внешней базой данных, которая по номеру карты и пин-коду сообщает размер доступной суммы, и в которую отправляются сведения о проведённых операциях.

### 4.2.3. Автомат выдачи

Нужно спроектировать устройство для управления автоматом выдачи каких-либо товаров. Автомат имеет фиксированный набор доступных товаров и их цен. Устройства ввода — устройство распознавания введённых купюр и клавиши управления. Устройства вывода — механизмы выдачи товара и сдачи.

### 4.2.4. Система управления лифтами

Нужно спроектировать устройство для управления движением нескольких лифтов (грузовых и пассажирских) в одном подъезде. С каждого этажа и из кабин лифтов к устройству подводится множество проводов от кнопок управления, а от устройства управления к этажам и кабинам идут провода, несущие информации о текущем положении лифта. Вызовы от пассажиров должны ставиться в очередь, а кабины должны выбирать оптимальный маршрут движения, но при этом не создавая неудобств для пассажиров.

При проектировании такой системы нужно уделить особое внимание графическому оформлению в Logisim: движение лифтов и их заполнение, а также наличие пассажиров на этажах должны наглядно отображаться.

### 4.2.5. Дозиметр

Дозиметр — довольно простое устройство. Оно считывает количество «щелчков» (импульсов), поступающих за секунду от входящего в его состав счётчика Гейгера, переводит это значение в общепринятые единицы измерения, и отображает на индикаторе. Случайность импульсов можно смоделировать, воспользовавшись компонентом «генератор случайных чисел», добавив возможность «регулировать» уровень ионизирующего излучения (то есть имитировать попадание устройства в разные внешние условия). Для усложнения задачи можно добавить память на несколько значений.

### 4.2.6. Бортовая система автомобиля

Нужно смоделировать электронную «начинку» современного

автомобиля. Устройство управления получает информацию с большого числа датчиков: уровень бензина, уровень масла, заряд аккумулятора, обороты двигателя, текущая передача, состояние дверей салона, капота и багажника, температура в салоне, и многое другое. Система должна выводить всю эту информацию водителю в удобочитаемом виде, позволяя управлять подсистемами автомобиля, и блокируя недопустимые (опасные) действия.

### 4.2.7. Часы-будильник

Нужно спроектировать стандартные часы-будильник, отображающие время с помощью семисегментных индикаторов. Нужно предусмотреть возможность установки времени часов и звонка будильника. Очевидно, что скорость течения времени для таких часов будет определяться параметром «Тактовая частота» из меню «Моделировать» Logisim.

### 4.2.8. Микроволновая печь

Нужно спроектировать устройство управления микроволновой печью. Пользователь с помощью панели управления задаёт режим и продолжительность работы, а устройство управления выводит нужную информацию на индикатор и управляет излучателем, приводом печи, а также блокировкой дверцы.

### 4.2.9. Стиральная машина

Нужно спроектировать устройство управления стиральной машиной. Пользователь с помощью панели управления задаёт режим и продолжительность стирки, а устройство управления выводит нужную информацию на индикатор и управляет всеми механическими устройствами машины: двигателем, нагревателем, блокиратором крышки, клапанами подачи воды и слива.

### 4.2.10. Кондиционер

Нужно спроектировать устройство управления кондиционером воздуха. Оно принимает от датчиков информацию о текущей температуре воздуха внутри и снаружи помещения, а также заданное пользователем желаемое значение температуры. Сравнивая эти значения, устройство управления подаёт управляющий сигнал на нагреватель или охладитель воздуха, пока температура в помещении не достигнет желаемой. Для демонстрации правильной работы устройства необходимо спроектировать отдельную схему, моделирующую реакцию окружающей среды (воздуха в помещении) на работу кондиционера.

### 4.2.11. Велоспидометр

Велоспидометр получает от датчика импульсы при каждом обороте колеса, и зная интервал времени и диаметр колеса, вычисляет текущую скорость движения, среднюю скорость на всём заезде, пройденный путь за заезд, и общий пройденный путь. Устройство позволяет выбирать отображаемую информацию, а также устанавливать диаметр колеса, сбрасывать статистику текущего заезда и общую статистику. Учтите, что это устройство работает не с целыми числами, а с вещественными (с плавающей точкой). В Logisim нет готовых компонентов для арифметических операций над вещественными числами, поэтому придётся либо проектировать их самостоятельно, либо каким-то образом приспособить для этого устройства компоненты, работающие с целочисленной арифметикой.

### 4.2.12. Машина Поста

Машина Поста — абстрактная вычислительная машина, предложенная Эмилем Леоном Постом. Изложение принципов её устройства выходит за рамки этого пособия. Однако проектирование в Logisim устройства, которое моделировало бы работу машины Поста — весьма интересная задача. Вместо ленты может выступать компонент ОЗУ, а программа загружается в виде образа памяти в другой компонент ОЗУ. Кроме того, нужно разработать формат представления команд машины Поста в шестнадцатеричном виде.

### 4.2.13. Клеточные автоматы

*Клеточный автомат* — набор клеток, образующих некоторую периодическую решетку с заданными правилами перехода, определяющими состояние клетки в следующий момент времени. Как правило, рассматриваются автоматы, где состояние в следующий момент времени определяется текущим состоянием самой клетки и её ближайших соседей. Возможно реализовать отдельную клетку автомата в виде отдельной схемы в Logisim. Соединяя большое количество таких подсхем (без проводов, непосредственным контактом), можно получить клетчатое поле. Если поверх каждой подсхемы расположить устройство отображения (например, светодиод), то можно наблюдать за потактовым изменением состояний клеток этого клеточного автомата.

Наиболее интересные для реализации клеточные автоматы (подробную информацию о правилах каждого из них несложно найти самостоятельно):

- Игра «Жизнь» Джона Конвея

- «Wireworld» Брайана Сильвермана
- «Муравей Лэнгтона»
- Элементарные клеточные автоматы (в частности, «правило 30», предложенное Стивеном Вольфрамом)

### 4.2.14. Бегущая строка

Нужно спроектировать устройство управления «бегущей строкой» — светодиодной матрицей, на которой может быть отображена строка символов, перемещающихся справа налево. Сам текст строки может быть сохранён в ОЗУ, изображение каждого отдельного символа (или даже отдельного столбца символа) формируется комбинационным устройством (как правило, собранным на основе ПЗУ), преобразующим код символа в комбинацию пикселей, а вся строка отображается на матрице посредством развёртки. Для хранения отображаемого в данный момент изображения, а также для реализации возможности поочерёдного добавления нового столбца и смещения всех столбцов на одну позицию влево, нужно использовать компонент «Сдвиговый регистр» из библиотеки «Память». Максимальная ширина матрицы в Logisim — 32 пикселя, поэтому для этой задачи нужно будет скомбинировать несколько матриц в одну.

### 4.2.15. Игры

Существует большое количество видеоигр, которые можно реализовать на простых цифровых устройствах без использования микропроцессора (то есть на «жёсткой логике»). Устройством отображения для большей части из них может быть светодиодная матрица. В рамках данного пособия нецелесообразно подробно описывать правила различных игр. Ограничимся лишь перечислением некоторых из них:

- Pong
- BlackJack
- Покер
- Однорукий бандит
- Лабиринт
- Кости

### 4.2.16. Робот

Можно представить себе задачу, когда по клетчатому полю с препятствиями передвигается «робот», задача которого обходить препятствия. На вход управляющего устройства робота поступает

информация об окружающем пространстве, а с выхода управляющего устройства поступают сигналы на приводы, осуществляющие движение (разворот на месте, движение в разных направлениях). Отображение поля и положения робота может производиться на светодиодную матрицу. При решении этой задачи крайне важно реализовать отдельно (как разные подсхемы) управляющее устройство, датчики, приводы, окружающее пространство и устройство отображения состояния всей системы.

### 4.2.17. Калькулятор

Нужно спроектировать калькулятор, работающий только с целыми числами. Пользователь последовательно вводит с помощью кнопок числа и желаемые операции над ними, а калькулятор производит вычисления и выдаёт результат на индикатор.

### 4.2.18. Процессор

Одно из самых сложных устройств, которое можно реализовать в Logisim — это процессор. Первое, что нужно сделать — определиться с архитектурой проектируемого процессора: CISC или RISC, архитектура фон Неймана или гарвардская, с конвейером или без; выбрать разрядность инструкций, данных и адреса. Следующий шаг — разработать точный набор инструкций. Можно попытаться повторить набор инструкций какого-либо известного процессора (например, MOS Technology 6502 или Intel 8080), или разработать полностью оригинальный. Дальнейшие действия — собственно проектирование — очень большая и сложная тема. Если вы решили взяться за такую задачу, то должны уже представлять этот процесс хотя бы в общих чертах.

Одним из вариантов курсового проекта может быть существенная доработка процессора lilpM32 или его ассемблера.