В. И. Сидельников

М. О. Слюта

СИСТЕМЫ АВТОМАТИЧЕСКОЙ ЗАЩИТЫ

Учебно-методическое пособие

Санкт-Петербург
2023

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Санкт-Петербургский государственный университет промышленных технологий и дизайна»
Высшая школа технологии и энергетики

В. И. Сидельников

М. О. Слюта

СИСТЕМЫ АВТОМАТИЧЕСКОЙ ЗАЩИТЫ

Учебно-методическое пособие

Утверждено Редакционно-издательским советом ВШТЭ СПбГУПТД

Санкт-Петербург
2023

УДК 620.1:681.3(075)
ББК 32.965я7
С 347

Рецензенты:
кандидат технических наук, доц. кафедры автоматизации технологических процессов и производств Высшей школы теплоэнергетики Санкт-Петербургского государственного университета промышленных технологий и дизайна
С. Л. Горобченко
доктор технических наук, профессор СПбГТИ
Л. А.Русинов

Сидельников, В. И.
С 347 Системы автоматической защиты : учебно-методическое пособие / В. И. Сидельников, М. О. Слюта. — СПб.: ВШТЭ СПбГУПТД, 2023. — 44 с.

Учебно-методическое пособие соответствует программе и учебному плану дисциплины «Системы автоматической защиты» для студентов, обучающихся по направлению подготовки 27.03.04 «Управление в технических системах».

Учебно-методическое пособие предназначено для подготовки бакалавров всех форм обучения.

УДК 620.1:681.3(075)
ББК 32.965я7

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Системы автоматической защиты (САЗ) и противоаварийной автоматической защиты (ПАЗ) являются обязательными составляющими, входящими в систему управления оборудованием и технологическим процессом. Системы защиты исторически возникли как наборы защитных блокировок, приводящих к остановке оборудования или переводу технологического процесса в безопасное состояние при выходе его параметров за предельно допустимые значения.

Любая система автоматической защиты состоит из трех основных функциональных частей:

− датчиков, измеряющих величину опасных параметров;

− исполнительных устройств, ликвидирующих аварийную ситуацию или приводящих параметры технологического процесса к нормальному уровню;

− логических устройств, принимающих сигналы и координирующих действия системы автоматического и программного управления, исполнительных устройств и сигнализации.

Логическое устройство представляет собой устройство дискретного действия, назначение которого – выработка правильной команды исполнительным органом в зависимости от состояния сигналов, поступивших на вход. Эти устройства составляются из так называемых логических элементов, каждый из которых реализует элементарную логическую операцию. Вся же совокупность логических элементов, входящих в автоматическую систему, производит сложное логическое действие, в результате которого на выходе системы появляется нужный сигнал.

В результате операции, выполняемой логическим элементом на его выходе, появляется сигнал «ДА» или «НЕТ» без промежуточного значения. Это действие может быть выражено двоичным кодом, в котором используются только две цифры: 0 (нет) и 1 (да).

Техническая реализация этих элементов может быть выполнена в различном виде: электромеханическом (в виде электромагнитных реле), магнитном (в виде перемагничивающихся сердечников с прямоугольной петлей гистерезиса) или электронном (транзисторы, интегральные микросхемы), что будет показано ниже.

В 1847 году английский математик Джордж Буль в своем трактате «Математический анализ логики» заложил основы современной алгебры логики, связав ее с логикой высказываний. Он ввел свою алгебраическую систему, которая содержала следующие функции: конъюнкция (алгебраическое умножение, оператор «AND»), дизъюнкция (алгебраическое сложение, оператор «OR») и инверсия (логическое отрицание, оператор «NOT»).

Впоследствии данная алгебра была названа Булевой. Упрощение формул в Булевой алгебре производится на основе эквивалентных преобразований, что позволяет для описания функционирования устройств использовать формальные языки и основанные на них формальные методы проектирования. Исследования

с помощью данного подхода позволяют достигать многовариантности в решении прикладных задач и возможность оптимального выбора схемотехнических решений.

Таким образом появляется возможность проектирования и исследования схем автоматической защиты с использованием математических методов при решении задач синтеза и анализа схем. Анализ схем с помощью алгебры логики позволяет найти функцию, описывающую работу заданной схемы. При этом каждому функциональному элементу схемы ставится в соответствие логическая операция или функция, что обеспечивает однозначное соответствие между элементами схемы и ее математическим описанием. Задачу синтеза можно сформулировать следующим образом: при заданных входных перемещениях и известной выходной функции спроектировать устройство, которое реализует эту функцию.

1. ЛОГИЧЕСКИЕ ФУНКЦИИ И ЛОГИЧЕСКИЕ ЭЛЕМЕНТЫ

1.1. Логические элементы

В качестве базового понятия, позволяющего осуществлять проектирование и исследование схем электронных устройств систем автоматической защиты, выступает логический элемент.

Логический элемент – это устройство, реализующее ту или иную логическую функцию.

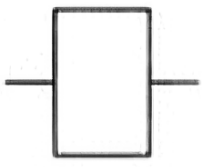
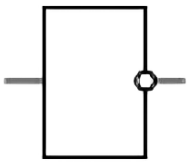
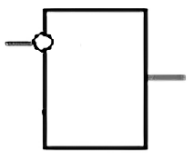
Логическая функция – это функция логических переменных, которая может принимать только два значения: 0 или 1. В свою очередь, сама логическая переменная (аргумент логической функции) тоже может принимать только два значения 0 или 1.

На практике логические элементы являются базовыми элементами цифровых схем, выполняющими элементарную логическую операцию (например: И, ИЛИ, НЕ, НЕ-ИЛИ, НЕ-И и другие). Каждый логический элемент имеет свое условное обозначение, которое выражает его логическую функцию, но не указывает на то, какая именно электронная схема в нем реализована. Это упрощает запись и понимание сложных логических схем. Так, для изображения логических вентилей всегда используются условные графические обозначения. В настоящее время в мире существует несколько общепринятых стандартов условных обозначений. Отечественной стандарт − ГОСТ 2.743-91, американский стандарт US ANSI 91-1984, стандарт, созданный Международной Электротехнической Комиссией, IEC 60617-12: 1997.

В таблице 1 представлены графические обозначения логических элементов этих трех стандартов.

Логический элемент на схемах изображают прямоугольником (как показано на рисунке 1).

Таблица 1 − Типы логических элементов относительно операции инверсии

Название	Логический элемент с простым входом	Логический элемент с инверсным выходом	Логический элемент с инверсным входом
Графическое обозначение			

Линии, которые находятся с левой стороны этого прямоугольника, называются входами, а с правой − выходами элемента. Внутри самого прямоугольника изображен указатель логической функции, которую выполняет данный элемент.

При описании работы логических элементов выходным сигналам ставят в однозначное соответствие функции, а входным сигналам – аргументы этих функций. В таблице 1 приведен пример обозначения логических элементов относительно операции инверсии.

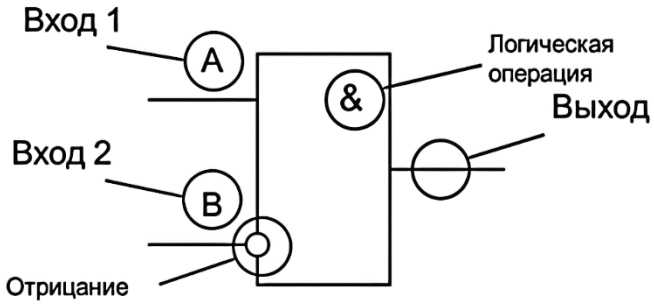


Рисунок 1. Схема изображения логического элемента

Целью решения задач анализа и синтеза схем, реализующих системы автоматической защиты, ниже приведена таблица 2 (обозначения и реализации логических элементов, их графическое изображение с учетом принятых обозначений).

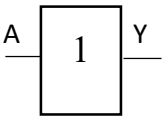
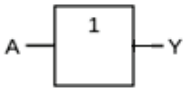
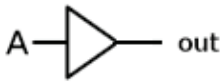
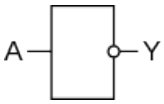
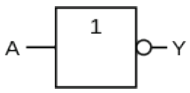
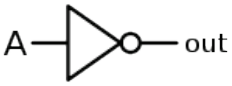
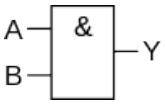
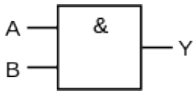
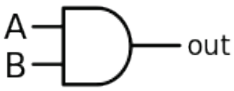
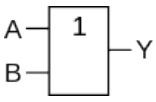
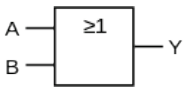
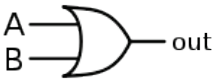
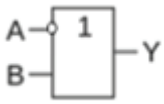
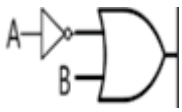
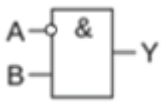
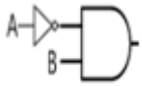
Исходя из таблицы 2, все логические элементы можно условно разделить на базовые, дополнительные и универсальные элементы.

К базовым элементам относятся три базовых элемента, реализующие функции (НЕ, И, ИЛИ), с использованием которых можно реализовать все другие логические операции. Для логических функций ИЛИ, И, НЕ существуют типовые технические схемы, которые были реализованы на реле, электронных лампах, дискретных полупроводниковых элементах и в настоящее время в интегральных схемах.

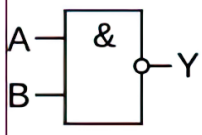
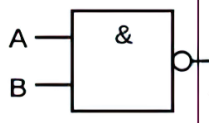
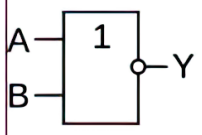
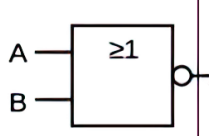
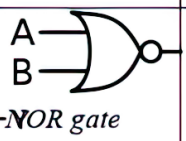
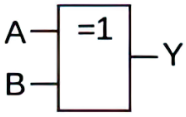
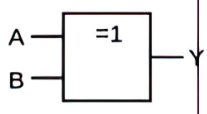
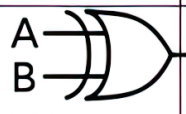
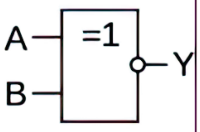
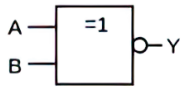
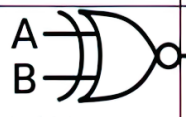
К дополнительным элементам относятся логические элементы ─ «повторители», так же имеющие один вход и один выход, но выходной сигнал повторяет значение входного сигнала. Такие элементы используются для «развязки» выходов логических элементов и для повышения их нагрузочной способности: Сумматор по mod2, реализующий функцию «Исключающее ИЛИ», Сумматор по mod2 с инверсным выходом, реализующий функцию «Исключающее ИЛИ с инверсией», Импликатор, Вентиль запрета.

Для систем автоматической защиты часто применяют «Вентили запрета», которые получили такое название потому, что сигнал по одному из входов «запрещает» либо «разрешает» прохождение на выход элемента сигнала, поданного на второй вход. Поэтому один вход называется входом запрета – он инверсный, а второй вход называют «информационным».

Таблица 2 − Основные логические элементы

Логический вентиль	**Условные графические обозначения**			**Функция**	**Таблица истинности A B Y**
	ГОСТ 2.743-91	**IEC 60617-12 : 1997**	**US ANSI 91-1984**		
Буфер Повторитель				Функция повторения Y=A	0 0 1 1
Инвертор **НЕ** (*NOT gate*)				Отрицание $Y = \overline{A}$ $Y = \neg A$	0 1 1 0
Конъюнктор **И** (*AND gate*)				Конъюнкция $Y = A \wedge B$ $Y = A \cdot B$ $Y = A \& B$ $Y = AB$	0 0 0 0 1 0 1 0 0 1 1 1
Дизъюнктор **ИЛИ** (*OR gate*)				Дизъюнкция $Y = A \vee B$ $Y = A + B$	0 0 0 0 1 1 1 0 1 1 1 1
Импликатор				Импликация $Y = A \rightarrow B$ $Y = \overline{A} + B$	0 0 1 0 1 1 1 0 0 1 1 1
Схема запрета входа A				Обратная импликация $Y = \neg(A \rightarrow B)$ $Y = \overline{A}B$	0 0 0 0 1 0 1 0 1 1 1 0

Окончание таблицы 2

Логический вентиль	**Условные графические обозначения**			**Функция**	**Таблица истинности**
	ГОСТ 2.743-91	**IEC 60617-12 : 1997**	**US ANSI 91-1984**		**A B Y**
Элемент Шеффера, **И-НЕ**				Штрих Шеффера $Y = A\|B$ $Y = \overline{A \wedge B}$	0 0 1 0 1 1 1 0 1 1 1 0
Элемент Пирса, **ИЛИ-НЕ**				Стрелка Пирса Y=A↓B $Y = \overline{A \vee B}$	0 0 1 0 1 0 1 0 0 1 1 0
Сумматор по mod2, **Исключающее ИЛИ**				Строгая дизъюнкция, Неравнозначность $Y = A \oplus B$ $Y = \overline{A}B + A\overline{B}$	0 0 0 0 1 1 1 0 1 1 1 0
Сумматор по mod2 с инверсным выходом, **Исключающее ИЛИ с инверсией**				Эквиваленция, Равнозначность Y=A~B; Y=A≡B $Y = \overline{A \oplus B}$ Y= $AB + \overline{A}\overline{B}$	0 0 1 0 1 0 1 0 0 1 1 1

Универсальные элементы представлены элементом Пирса, который реализует функцию «Срелка Пирса», позволяющий реализовывать в своих соединениях друг с другом все известные логические операции. Вентиль – элемент Шеффера, который реализует функцию «Штрих Шеффера», позволяющий реализовывать в своих соединениях друг с другом все известные логические операции.

1.2. Логические схемы

Для реализации сложных функций логические элементы объединяются в логические схемы, которые реализуются логическим устройством.

Модель логической схемы, как схемы элементов, реализующих логическую функцию, приведена на рисунке 2.

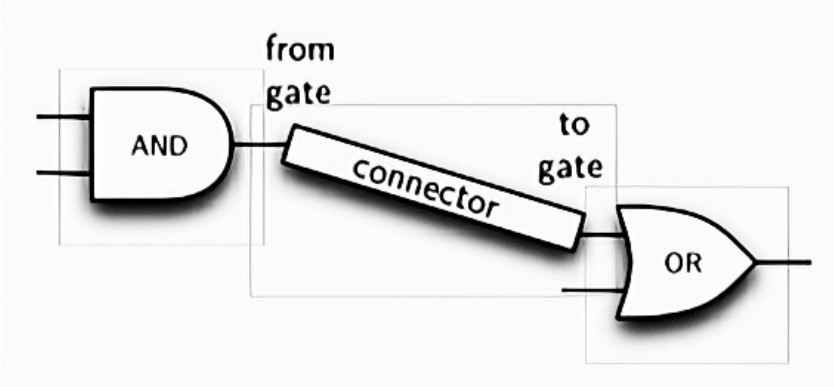
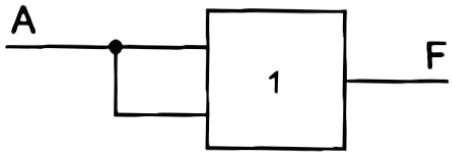
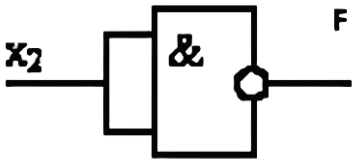
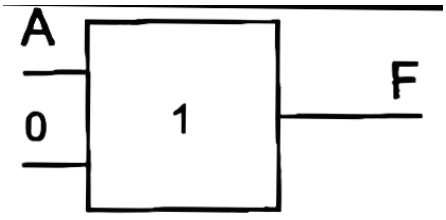


Рисунок 2. Модель логической схемы

При проектировании САЗ иногда возникают вопросы, связанные с применением нетиповых логических функций. В таблице 3 приведены примеры таких схем и формулы логических схем, которые они реализуют [1].

Таблица 3 – Примеры логических схем

Логическая схема	Логическая функция
	$F = A \vee A$
	$F = \neg(X2 \wedge X2)$
	$F = A \vee 0$

2. ПРИМЕРЫ РЕШЕНИЯ ЗАДАЧ

2.1. Переход от логической схемы к логической функции

Задача 1. По заданной схеме требуется определить функцию Y, реализующуюся схемой, приведенной на рисунке 3.

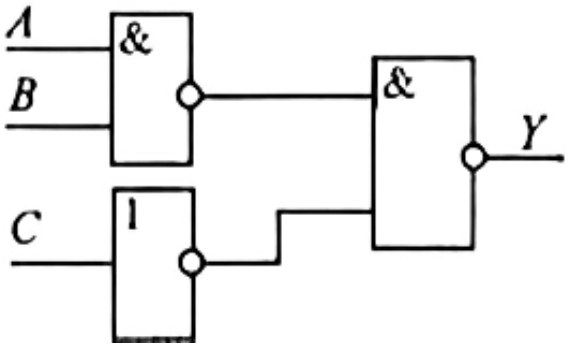


Рисунок 3. Заданная схема

Алгоритм решения

1. Подчитываются количество логических элементов (рисунок 4), входящих в схему (в данном случае их три):

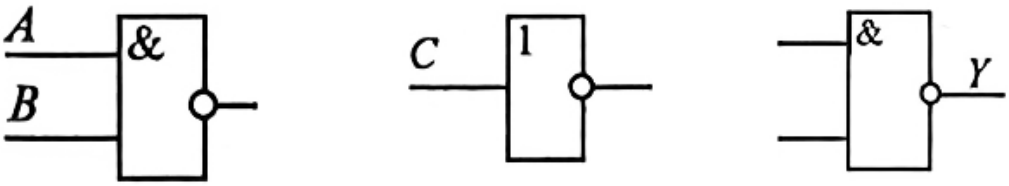


Рисунок 4. Логические элементы

2. Выходы каждого логического элемента обозначаются проиндексированными функциями в порядке, начиная от последнего, причем последний логический элемент имеет название конечной функции (Y1,Y2,Y).

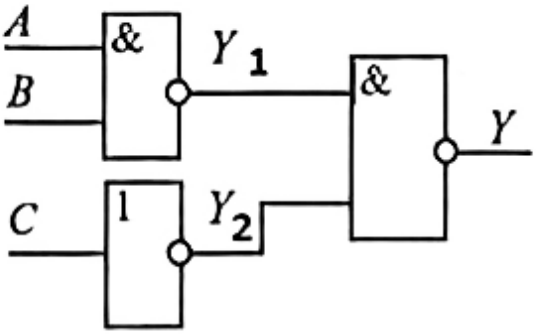


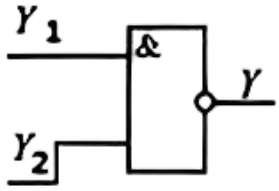
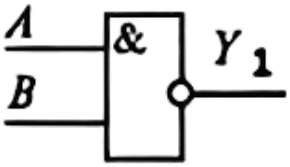
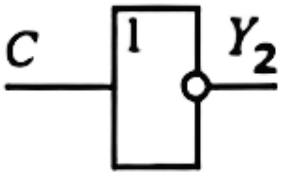
Рисунок 5. Обозначение элементов

3. Заполняем таблицу 4 по следующему принципу: записываются выходные функции каждого элемента в виде формул в соответствии с введенными обозначениями, в порядке начиная с последнего.

4. Производится подстановка одних выходных функций через другие, используя входные переменные и логические функции, полученные в таблице 4.

$$Y=\neg(Y1\wedge Y2)=\neg(\neg(A\wedge B)\wedge(\neg C))$$

Таблица 4 – Логические элементы и функции

Логические элементы	Логические функции
	Y = ¬(Y1 ∧ Y2)
	Y1 = ¬(A∧B)
	Y2 = ¬C

5. Записывается получившаяся булева функция через входные переменные:

$$Y=\neg(\neg(A\wedge B)\wedge(\neg C))$$

2.2. Построение логической схемы, соответствующей логическому выражению

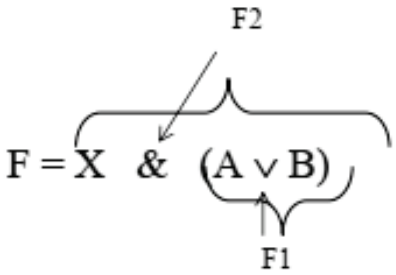
Задача 2. Составить логическую схему для логической функции:

$$F = X \,\&\, (Y \vee Z).$$

Алгоритм решения

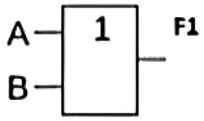
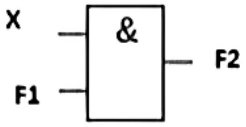
1. Определяем число логических переменных и их названия (3 переменных: X, Y, Z), т.е. у нас будет три входа: X, Y, Z.

2. Определяем приоритеты выполнения логических операций для исходной функции с учетом существующих логических элементов и соответствующие им логические функции (табл. 3) и даем им соответствующие названия проиндексированных функций.



3. Заполнить таблицу 5 по следующему принципу: изобразить для каждой логической операции и полученной проиндексированной функции соответствующий ей вентиль в порядке приоритета.

Таблица 5 – Логические элементы и функции

Приоритет	Функция	Логический элемент
1	F1=A v B	
2	F2= X & F1	
Итого	F= F2	?

4. Соединить элементы в порядке одинаковых входов и выходов. В данном случае стыковка вентилей по функции F1 (рис. 6).

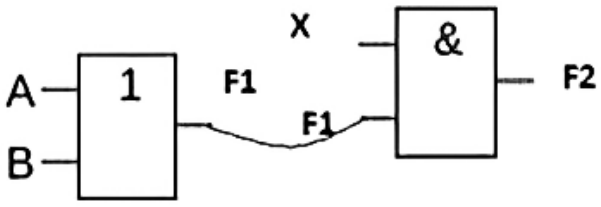


Рисунок 6. Элементы в порядке одинаковых входов и выходов

5. В итоге получили логическую схему, представленную на рисунке 7.

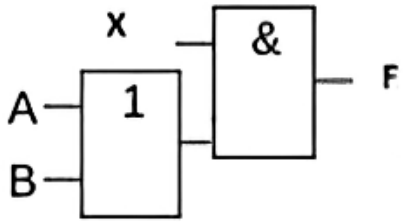


Рисунок 7. Логическая схема

Задача 3.

Построить логическую схему, соответствующую логическому выражению:

$$F = X \,\&\, Y \vee (\neg(Y \vee X)).$$

Алгоритм решения

1. Определить число логических переменных и их названия.

Используются две логические переменные: X и Y, каждая используется дважды. Это означает, что у нас будет два входа с названиями X, Y (рис. 8).

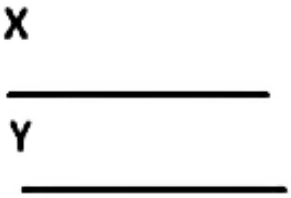
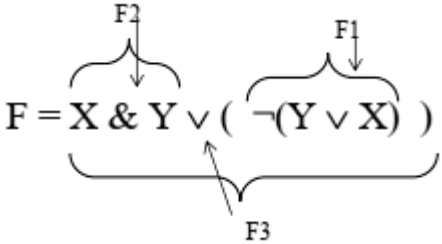


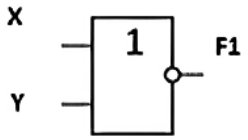
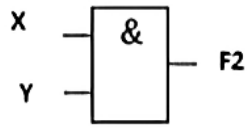
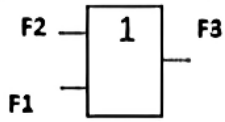
Рисунок 8. Два входа с названиями X, Y

2. Определим приоритеты выполнения логических операций для исходной функции с учетом существующих логических элементов и соответствующие им логические функции и даем им соответствующие названия проиндексированных функций. В нашем случае часть формулы: $\neg(Y \vee X)$ рассматривается как единое целое, так как она соответствует Элементу Пирса (табл. 3).



3. Заполняется таблица 6, где для каждой логической операции и полученной проиндексированной функции соответствует элемент в порядке приоритета.

Таблица 6 – Логические элементы и функции

Приоритет	Функция	Логический элемент
1	F1=(¬(Y v X))	
2	F2= X & Y	
3	F3= F2 v F1	
Итого	F= F3	

4. Соединяем элементы в порядке одинаковых входов и выходов. В данном случае стыковка элементов по функции F1, F2. X, Y – одинаковые (рис. 9).

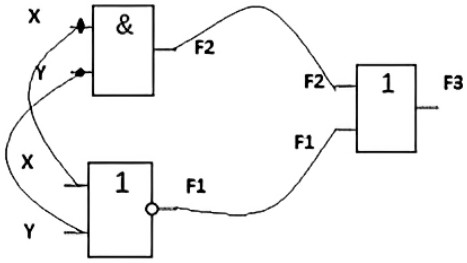


Рисунок 9. Элементы в порядке одинаковых входов и выходов

5. В итоге получили логическую схему (рис. 10).

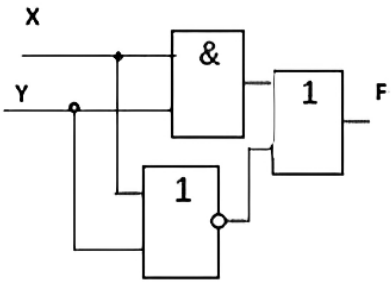


Рисунок 10. Логическая схема

3. ЭКВИВАЛЕНТНОСТЬ ЛОГИЧЕСКИХ СХЕМ

Эквивалентность логических схем. Две логические схемы называются эквивалентными, если они выполняют одинаковые функции. Факт эквивалентности двух схем устанавливается по их таблицам истинности. Если таблицы истинности совпадают, значит, схемы эквивалентны.

Таблица истинности для логических схем – это табличное представление логической схемы, в котором перечислены все возможные сочетания значений истинности входных сигналов (операндов) вместе со значением истинности выходного сигнала (результата операции) для каждого из этих сочетаний.

Свойство эквивалентности логических схем активно используется в больших интегральных схемах с целью унификации их структуры, синтез логических и вычислительных схем выполняется на базе только одного из универсальных вентилей. Каждый из этих вентилей структурно легко реализуется на основе базовых, и, наоборот, каждый базовый вентиль конструируется из универсальных. Ниже приведена таблица 7, где представлены примеры конструирования одних элементов из других.

Логические элементы и другие цифровые электронные устройства выпускаются в составе серий микросхем.

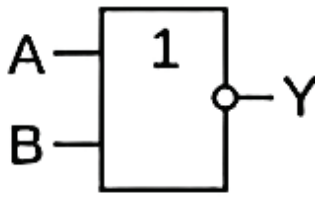
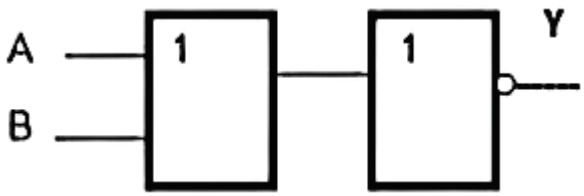
В современных интегральных схемах целью большей унификации в качестве базовой логической схемы используется всего одна из схем, например:
– НЕ-И – (NAND, элемент (штрих) Шеффера);
– НЕ-ИЛИ (NOR, элемент (стрелка) Пирса).

Серия микросхем − это совокупность микросхем, характеризуемых общими технологическими и схемотехническими решениями, а также уровнями электрических сигналов и напряжения питания [2].

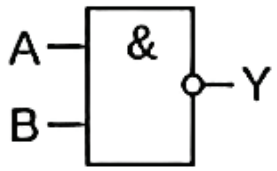
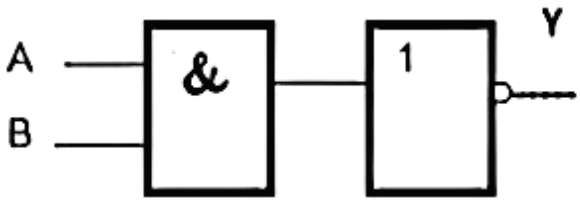
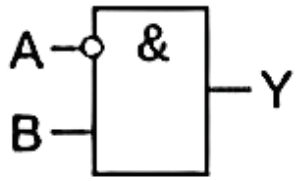
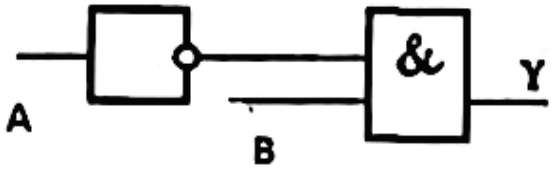
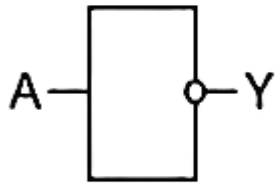
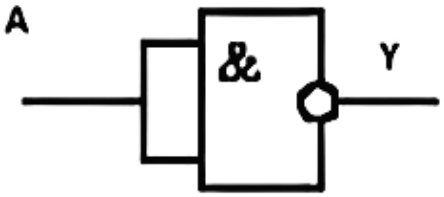
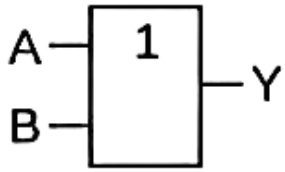
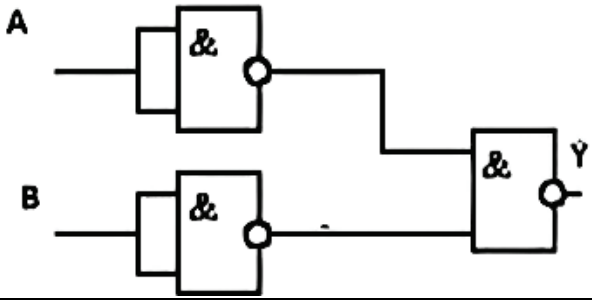
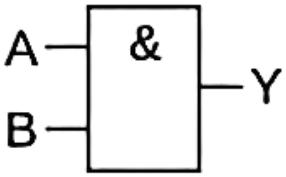
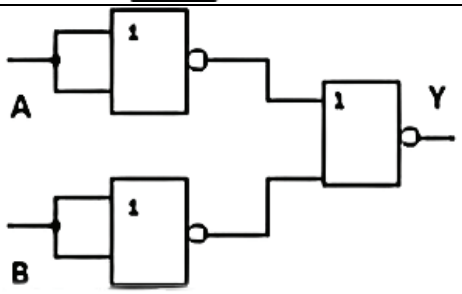
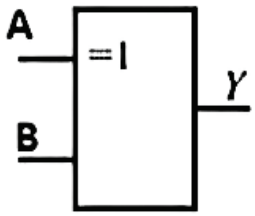
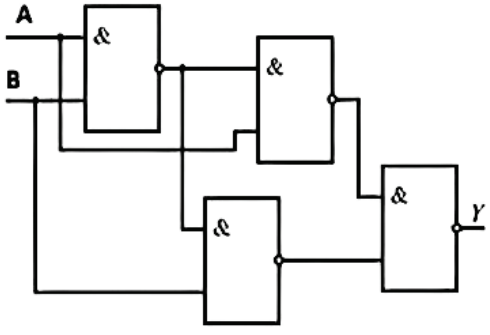
Пример построения логических схем с помощью соединения микросхем представлен на рисунке 11.

Ниже приводятся примеры выполнения лабораторных работ по синтезу и анализу логических схем, связанных с созданием систем автоматической защиты с использованием программных средств.

Таблица 7 − Примеры эквивалентных схем для логических элементов

Примечание	Логический элемент	Эквивалентная схема логическому элементу
Реализация Элемента Пирса с помощью функций ИЛИ, НЕ		

Окончание таблицы 7

Примечание	Логический элемент	Эквивалентная схема логическому элементу
Реализация Элемента Шеффера с помощью функций И, НЕ		
Реализация функции запрета (обратной импликации) с помощью функций НЕ, И		
Реализация функции НЕ с помощью Элемента Шеффера		
Реализация функции ИЛИ с помощью Элемента Шеффера		
Реализация функции И с помощью Элемента Пирса		
Реализация Сумматора по mod2 с помощью Элемента Шеффера		

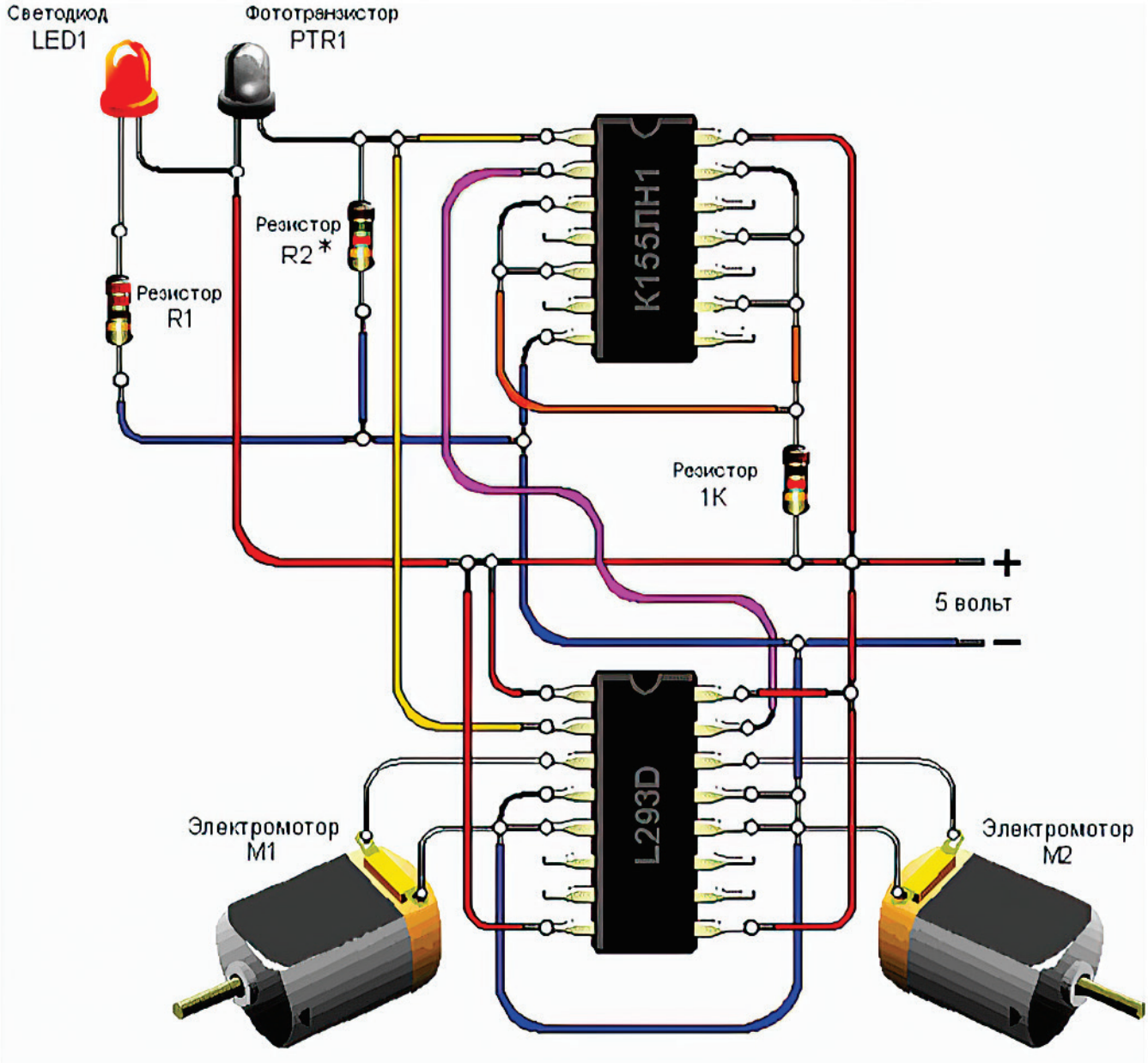


Рисунок 11. Построение логических схем с помощью соединения микросхем

4. ЛАБОРАТОРНЫЕ РАБОТЫ

В данном разделе представлены задания по лабораторным работам, которые студенты будут выполнять в программной среде Logisim.

Logisim – программное обеспечение для моделирования цифровых логических схем с помощью графического интерфейса (рис.12). Основной разработчик программы Carl Burch. Данное программное обеспечение, выпущенное под GNU GPL, является свободным для пользования. Код программы полностью написан на Java с использованием библиотеки Swing для графического интерфейса пользователя.

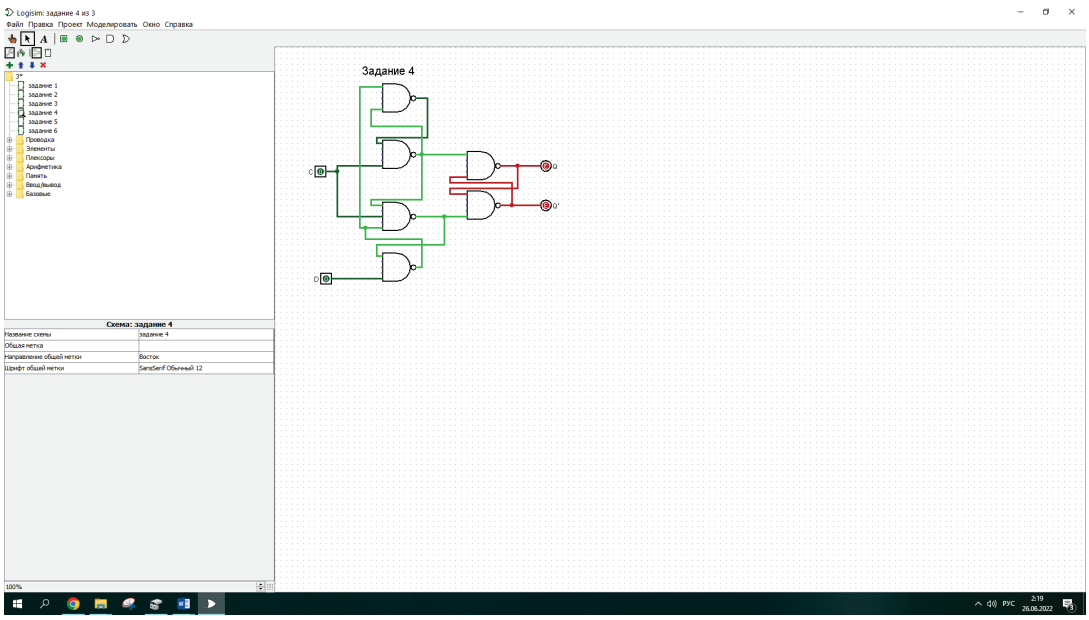


Рисунок 12. Интерфейс программного обеспечения Logisim

Данная программная среда может запускаться на Microsoft Windows, MacOS и Linux. Программа чаще всего используется для разработки и экспериментов с цифровыми схемами при моделировании. Схемы разрабатываются в Logisim с помощью графического интерфейса. В отличие от большинства других программ Logisim позволяет редактировать схемы в процессе моделирования. Относительная простота интерфейса делает программу удобной для пользования. Данная программная среда наполнена множеством инструментов, которые необходимы для работы с цифровыми логическими схемами: элементы ввода и вывода, логические элементы *НЕ, И, ИЛИ,* мультиплексоры, блоки арифметических операций, триггеры, элементы памяти [3].

Logisim распространяется с поддержкой русского интерфейса и полной документацией на русском языке, что очень удобно для пользователей.

ЛАБОРАТОРНАЯ РАБОТА № 1

Знакомство с программным обеспечением Logisim

Цель работы: познакомиться с интерфейсом программной среды Logisim, изучить основные инструменты данной программы.

Задание

Необходимо построить схему, представленную на рисунке 13, в программной среде Logisim.

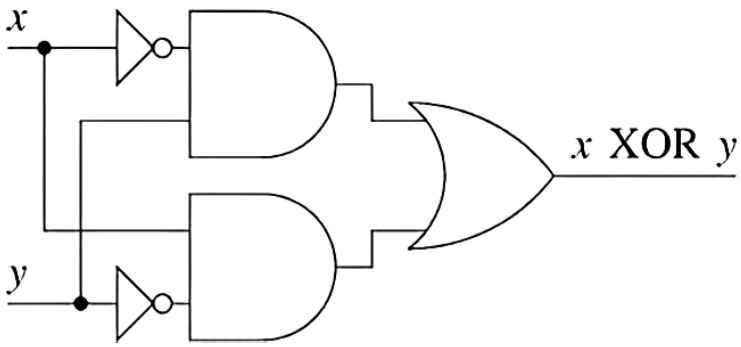


Рисунок 13. Схема для построения

Ход работы

1. На рабочем столе открываем программное обеспечение Logisim.

2. Для начала необходимо добавить два элемента *И* из библиотеки данной программы:

2.1. Для этого нажмите на инструмент *Элемент И* на панели инструментов (⊐, предпоследний инструмент в списке). Затем щёлкните в области редактирования там, где хотите поместить первый элемент.

2.2. Необходимо оставить свободное место для других элементов слева. Затем нажмите на инструмент *Элемент И* снова и поместите второй элемент *И* под первым.

2.3 Получается схема, представленная на рисунке 14.

3. Обратите внимание на пять точек на левой стороне элемента *И*. Это элементы для подсоединения проводов. В данной работе используется только два из них (для схемы Исключающее *ИЛИ*).

4. Теперь необходимо добавить следующие элементы. Сначала щёлкните на инструмент *Элемент ИЛИ* (▷), затем щёлкните там, куда хотите его поместить.

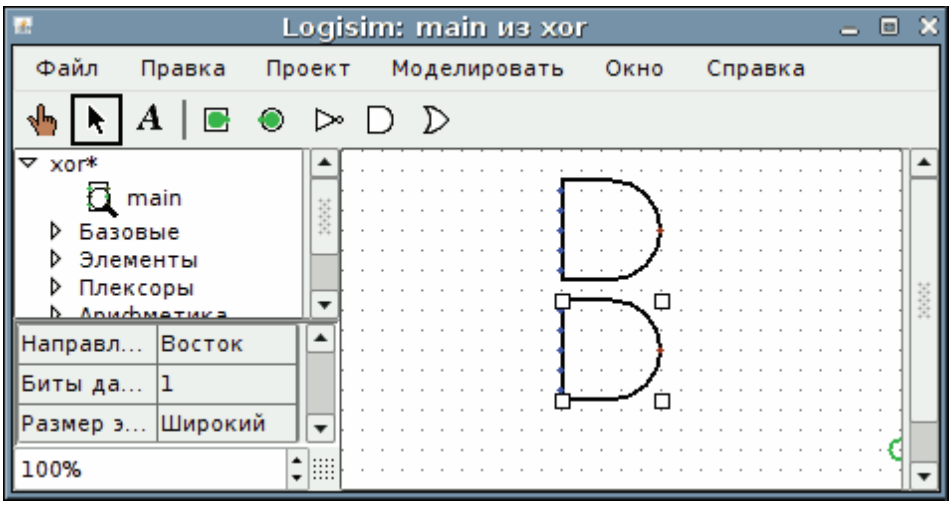


Рисунок 14. Элементы *И*

5. После этого расположите два элемента *НЕ* на холсте (на панели инструментов нажмите *Элемент НЕ* ( )) и поместите его на рабочее пространство и затем повторите действие для второго элемента *НЕ*. В результате проделанной работы получится набор из элементов, представленный на рисунке 15 [4].

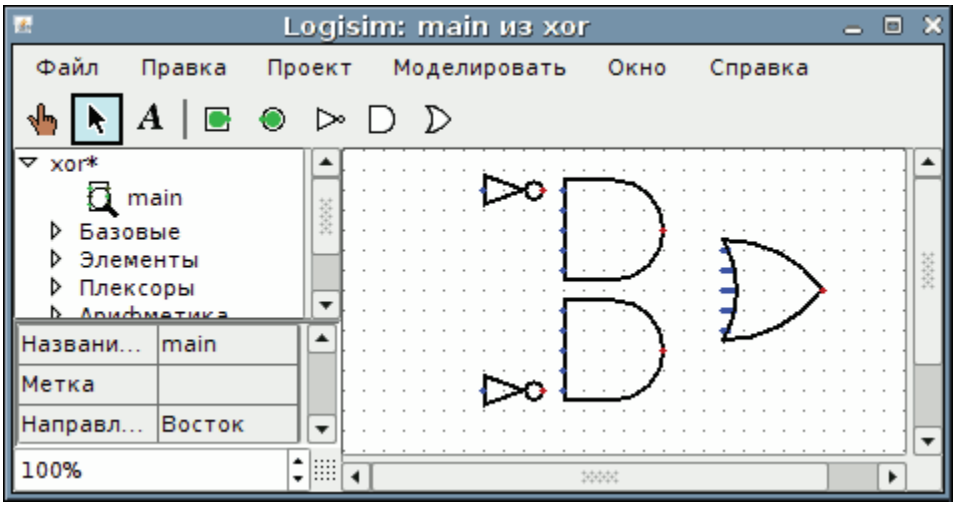


Рисунок 15. Элементы *НЕ, И, ИЛИ*

Необходимо располагать элементы ровно, друг за другом, чтобы схема была аккуратно построена и хорошо читалась. В дальнейшем элементы будут соединяться проводами, поэтому можно оставить немного пространства между ними.

6. После того как все необходимые элементы выбраны, следует добавить в чертёж два входа *x* и *y*. Выберите инструмент *Добавить входной контакт* ( ) и разместите контакты. Вам также нужно разместить выходной контакт рядом с выходом *Элемента ИЛИ*, используя инструмент *Добавить выходной контакт* ( ) (рис.16).

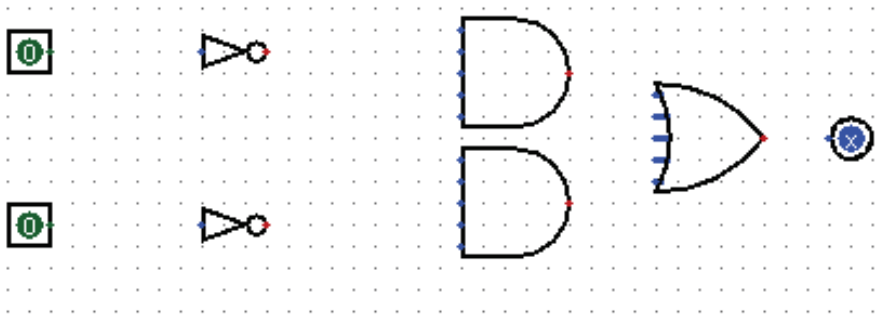


Рисунок 16. Элементы *НЕ, И, ИЛИ*, входной и выходной контакты

Для изменения местоположения объекта можно выбрать его с помощью инструмента *Правка* ( ) и перетащить в нужное место. Также можно удалить элемент полностью, выбрав *Удалить* из меню Правка или нажав клавишу Delete.

7. После того как все компоненты закреплены на рабочем пространстве, можно начинать добавление проводов. Для этого потребуется инструмент *Правка ( )*. Необходимо подвести курсор к точке, где подключается провод у элемента. При наведении курсора на данную точку, появляется зелёный кружок. Затем зажмите кнопку мыши и проведите провода к необходимым элементам.

7.1. Провода в Logisim должны быть горизонтальными или вертикальными. Чтобы соединить верхний вход с элементом *НЕ*, а затем с элементом *И*, нужно добавить три разных провода.

Logisim автоматически подключает провода к элементам и друг к другу. Когда вы рисуете провода, то можете увидеть несколько синих или серых проводов. Синий показывает, что значение в этой точке "неизвестно", а серый показывает, что провод не подключен ни к чему. В этом нет ничего особенного в процессе построения схемы. Но когда построение схемы будет закончено, то ни один из проводов не должен быть синего или серого цвета (только не присоединённые ответвления элемента *ИЛИ* останутся синими).

Если остались синий или серый провод, когда схема уже построена, значит что-то сделано не верно. Важно, чтобы подключение было правильным. Logisim отображает маленькие точки на компоненте, чтобы показать, куда подключать провода. Когда схема будет собрана, то вы увидите, что точки стали из синих светло- или тёмно-зелёными.

7.2. После того как будут подключены все провода, то они будут светло- или тёмно-зелёного цвета (рис.17).

8. Для того чтобы схема была проста для понимания и быстро читалась, то необходимо подписать некоторые ее элементы. Для этого выберите инструмент *Текст (A)*. Чтобы подписать входной контакт, то лучше нажать курсором непосредственно на входном контакте, чтобы при изменении в схеме набранный текст двигался вместе с контактом. Затем то же самое повторить для выходного контакта. По желанию можно подписать другие элементы, а также можно просто в любом свободном месте сделать необходимые для себя записи, например, указать название схемы (рис.18).

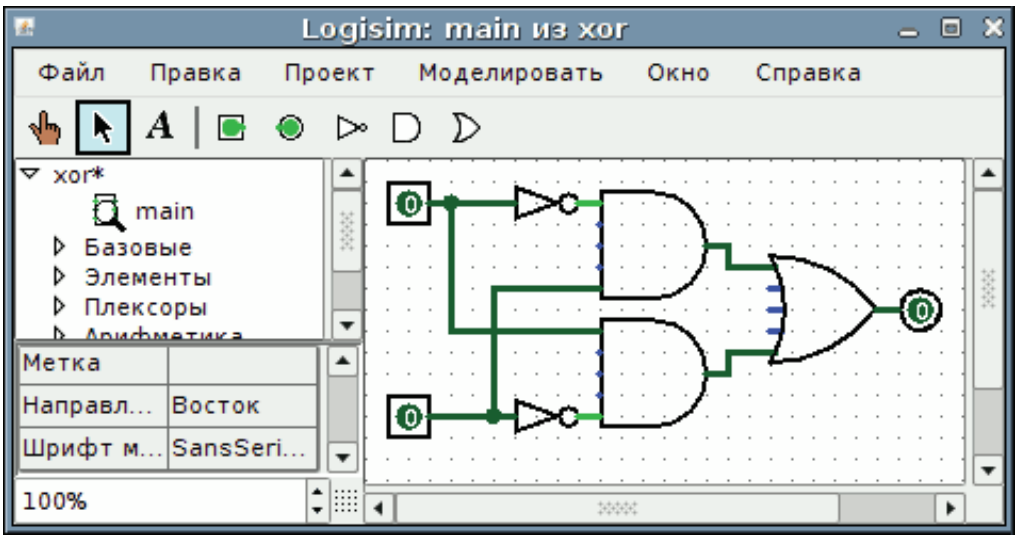


Рисунок 17. Подключение проводов в программе Logisim

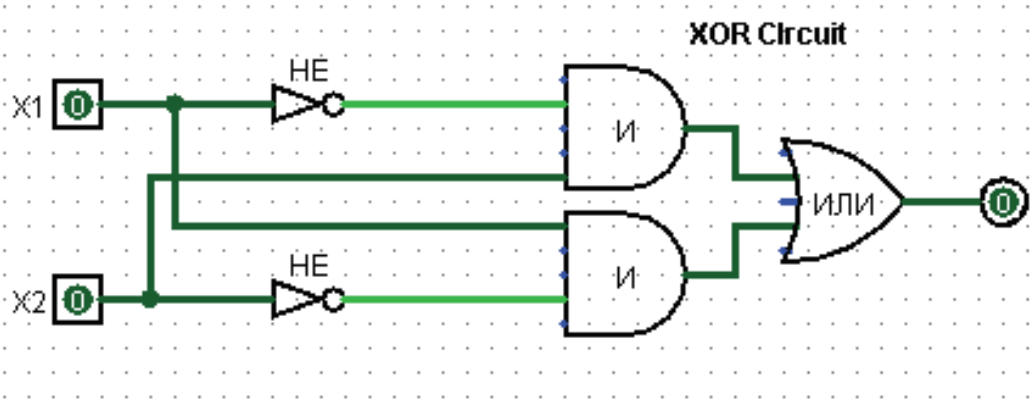


Рисунок 18. Готовая схема

9. Последний необходимый шаг в данной лабораторной работе − проверка схемы. Проверка нужна, чтобы удостовериться, что собранная схема действительно работает.

На рисунке 18 видно, что на обоих входных контактах нули, и на выходном контакте также. Это говорит о том, что схема вычисляет «0», когда на обоих входах «0».

Попробуем другую комбинацию входов. Выберите инструмент *Нажатие (*) и начните менять значения на входах, нажимая на них. Каждый раз, когда нажимаете на вход, его значение будет переключаться. Например, нажмите сначала на нижний вход (рис. 19).

Можно увидеть, что цвет проводов меняется в зависимости от входного значения. Если значение «1», то цвет будет светло-зелёный, иначе тёмно-зелёный. Также можно увидеть, что при изменении входного значения, выходное значение сменилось на «1».

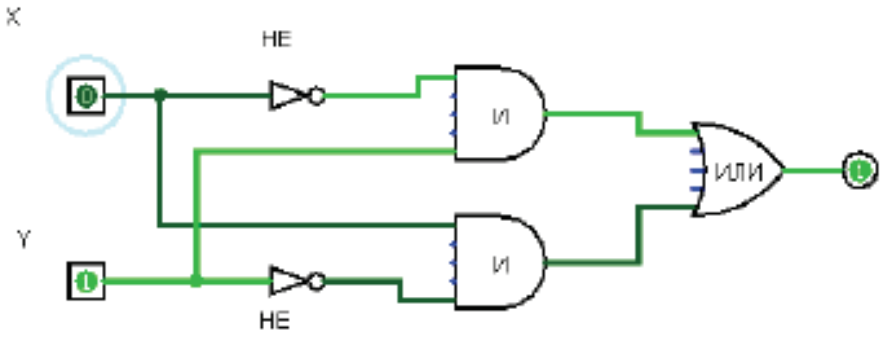


Рисунок 19. Схема при X=0, Y=1

10. Затем необходимо проверить остальные комбинации (рис. 20, рис. 21). Для этого в программе нужно вызвать таблицу истинности (рис. 22) и сверить входные и выходные значения.

11. После проверки можно убедиться, что все выходные значения соответствуют таблице истинности данной функции. Значит, схема работает корректно.

12. После выполнения данной лабораторной работы студенту необходимо сохранить файл и показать преподавателю. Затем подготовить отчет о выполненной работе, где будут прописаны цель работы, ход работы и выводы [4].

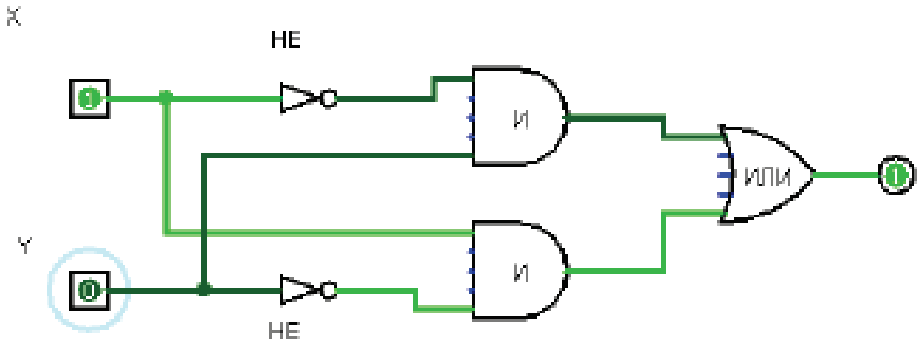


Рисунок 20. Схема при X=1, Y=0

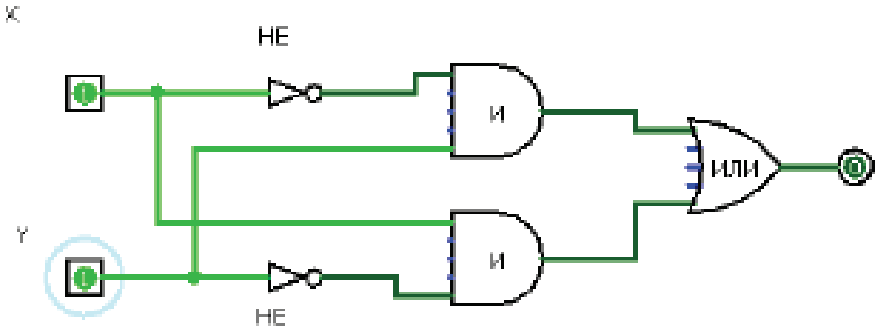


Рисунок 21. Схема при X=1, Y=1

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

Рисунок 22. Таблица истинности

ЛАБОРАТОРНАЯ РАБОТА № 2
Построение двоичного сумматора в программе Logisim

Цель работы:

Изучить правила выполнения арифметических действий над двоичными числами и исследовать принципы построения двоичных сумматоров.

Ход работы

Основным элементом, используемым в двоичных арифметических элементах, является полусумматор. Начнем с так называемой схемы сравнения:

1. Запустите программу Logisim.

2. С помощью «Панели инструментов» постройте схему сравнения (см. рис. 23).

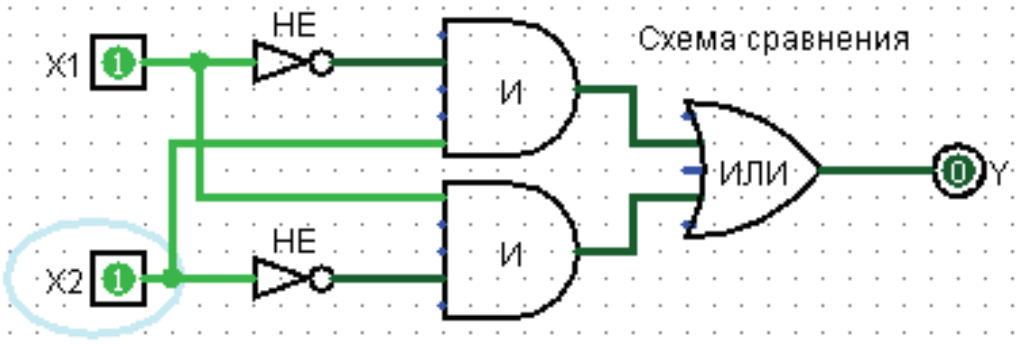


Рисунок 23. Схема сравнения

3. С помощью «Таблицы атрибутов» задать метки: вход X1 и X2, логические И, ИЛИ и НЕ, выход Y.

4. Проводник − Базовые − Инструмент Текст: Подписать схему, как «Схема сравнения».

5. С помощью инструмента «Изменять значения в схеме»  поэкспериментируйте с подачей на входы X1 и X2 логической единицы 1 и логического 0.

6. По команде: Проект − Анализировать схему, получить Таблицу истинности схемы сравнения (рис. 24).

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Рисунок 24. Таблица истинности

Схема сравнения получает младший разряд числа при сложении двух двоичных чисел (бит) без учета переноса! Например: 1 + 1 = 0 младший разряд, перенос 1.

7. С помощью инструмента «Добавить схему» ✚ добавим схему «Полусумматор» (рис. 25).

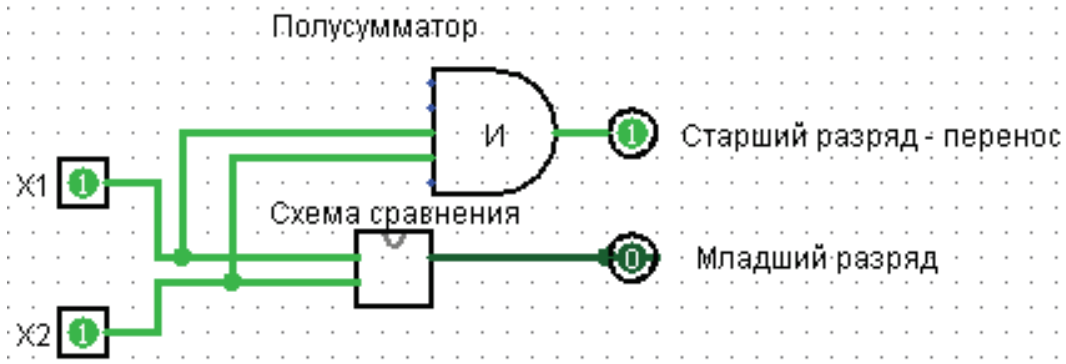


Рисунок 25. Полусумматор

8. Проект – Анализировать схему – получить Таблицу истинности полусумматора (рис. 26).

Схема, позволяющая складывать два двоичных числа (бит), называется полусумматором. В нашем случае P – перенос, S – младший разряд, остаток. Однако при сложении двух двоичных чисел недостаточно использовать полусумматор, т. к. полусумматор не имеет входа для учета переносов из других разрядов.

X1	X2	P	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Рисунок 26. Таблица истинности полусумматора

9. Добавить схему «Сумматор»

10. Используя подсхему «Полусумматор», построить «Сумматор» (рис. 27).

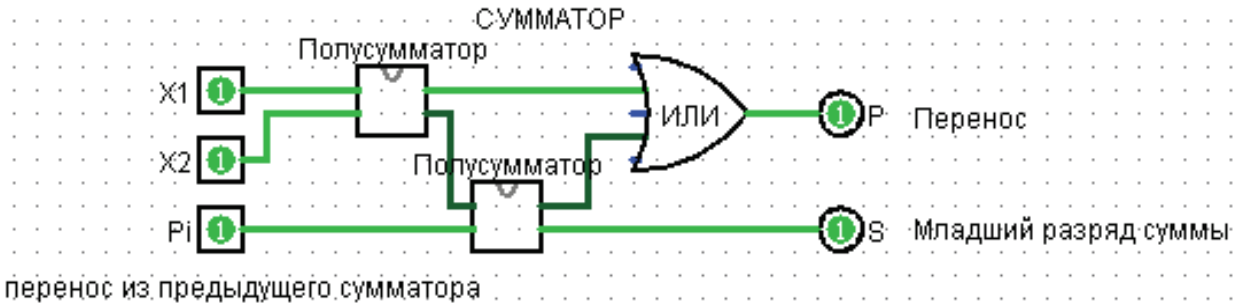


Рисунок 27. Сумматор

11. По команде: Проект – Анализировать схему получить Таблицу истинности «Сумматора» (рис. 28).

X1	X2	Pi	P	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Рисунок 28. Таблица истинности «Сумматора»

12. Построим схему из 4-х сумматоров, которые позволят складывать два четырехразрядных числа (рис. 29).

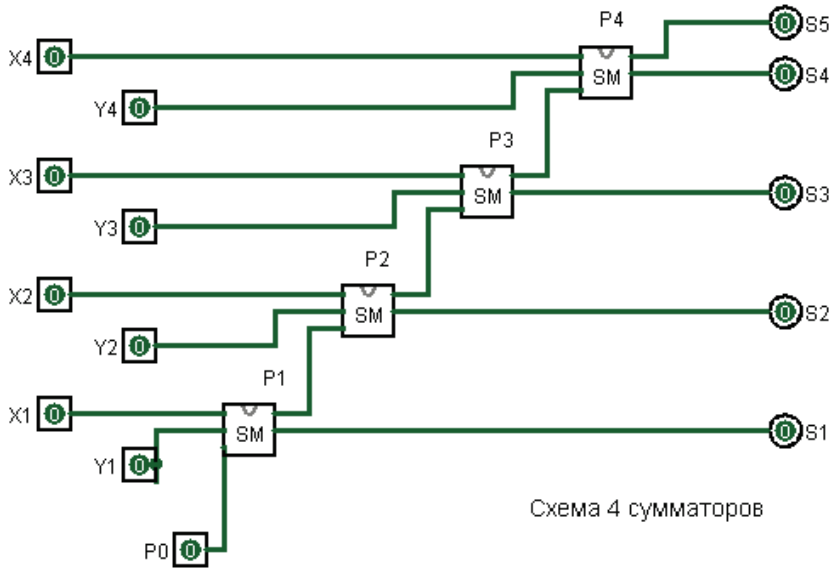


Рисунок 29. Схема из 4-х сумматоров

Где:

- X1 и Y1 слагаемые первого сумматора, X2 и Y2 – второго и т. д.;
- S1, S2, S3 .. S5 – младший разряд суммы;
- P1, P2, P3 и P4 – перенос, старший разряд сумматора 1, 2, 3 и 4;
- P0 – всегда равно 0, т. к. в первом сумматоре складываются первые двоичные числа X1 и Y1, переноса нет.

В нашем примере выполняем сложение двух четырехразрядных чисел: 0001 + 0001 = 00010

```
   X4 X3  X2  X1
+  Y4 Y3  Y2  Y1
S5 S4 S3  S2  S1
    0  0 0  1
+   0  0 0  1
0   0  0 0  0
```

13. Проверить схему при следующих входных данных:

```
  1 0 0 1 Xi
+ 0 0 1 1 Yi
0 1 1 0 0 Si
```

Самостоятельно построить схему, позволяющую складывать 8 разрядов двоичных чисел.

После выполнения данной лабораторной работы студенту необходимо сохранить файл и показать преподавателю. Затем подготовить отчет о выполненной работе, где будут прописаны цель работы, ход работы и выводы [4].

ЛАБОРАТОРНАЯ РАБОТА № 3

Элементарные устройства памяти

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 асинхронными и синхронными. В данном курсе будем работать только с синхронными устройствами.

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 (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 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 8.

Q' означает значение, противоположное значению, хранимому в триггере в данный момент.

Можно дать словесное описание поведения триггеров:

D-триггер: когда тактовый вход срабатывает, значение, хранящееся в триггере, мгновенно становится значением входа D (данные).

Таблица 8 − 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	?

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).

На схемах представлены различные реализации триггеров на логических элементах (рис. 30, рис. 31, рис. 32, рис. 33) На рисунке 33 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.

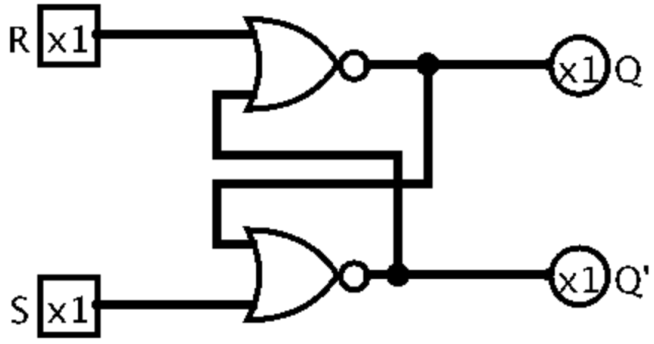


Рисунок 30. Асинхронный RS-триггер

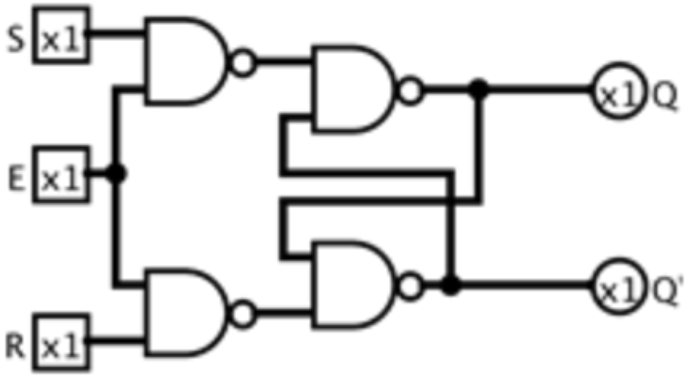


Рисунок 31. RS-триггер с разрешающим входом

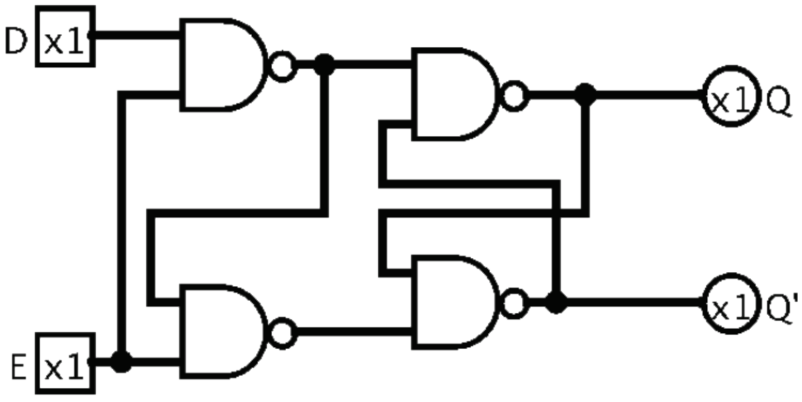


Рисунок 32. D-триггер с разрешающим входом

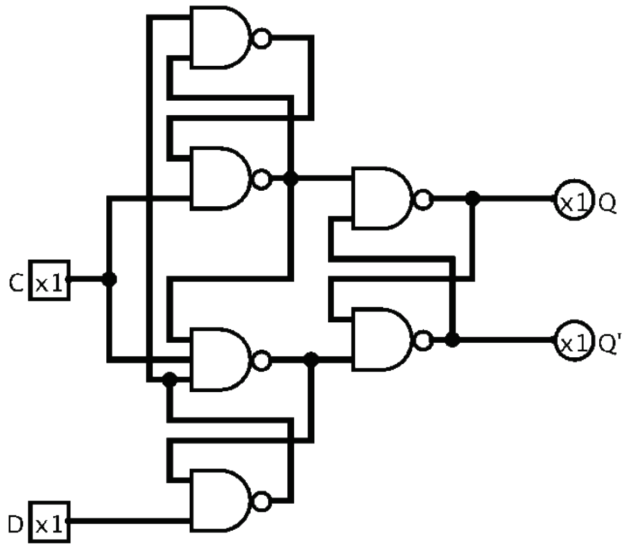


Рисунок 33. Синхронный D-триггер

Двухступенчатый синхронный D-триггер показан на рисунке 34. При поступлении на его тактовый вход переднего фронта обновится только состояние на выходе первой ступени, а обновление состояния всего триггера произойдёт при заднем фронте.

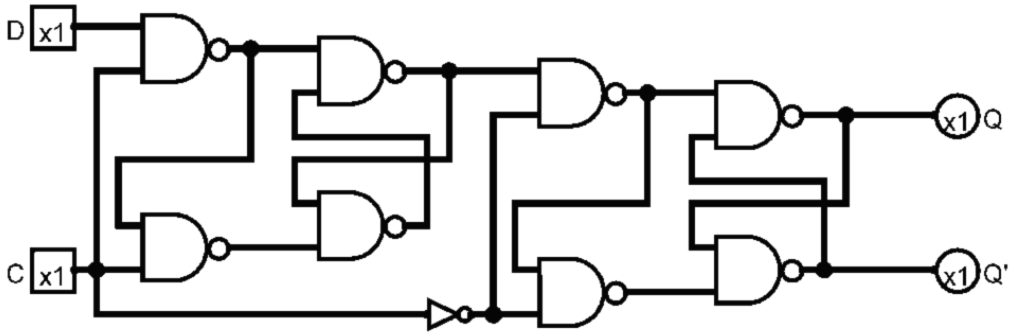


Рисунок 34. Двухступенчатый синхронный D-триггер

При моделировании триггеров на логических элементах в Logisim следует установить флажок «Добавить шум к задержкам компонентов» в окне параметров 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 [5].

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.

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.

После выполнения данной лабораторной работы студенту необходимо сохранить файл и показать преподавателю.

Для защиты данной работы студенту необходимо показать работу устройств в соответствии с таблицей истинности (если таблица не открывается, то воспользоваться калькулятором). Затем подготовить отчет о выполненной работе, где будут прописаны цель работы, ход работы и выводы [5].

ЛАБОРАТОРНАЯ РАБОТА № 4

Арифметика

Группу битов можно рассматривать как отдельные разряды многобитного значения, которое, в свою очередь, можно рассматривать как двоичное представление целого числа.

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 9.

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 (табл. 10).

Таблица 9 − 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	**101011**1 (-41)	**1010**100 (-44)

Таблица 10 − 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

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 «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. 35).

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 36. 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.

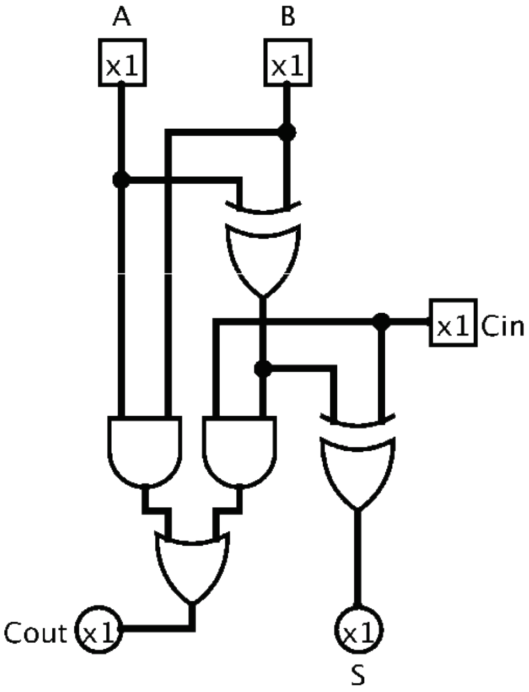


Рисунок 35. Полный одноразрядный сумматор

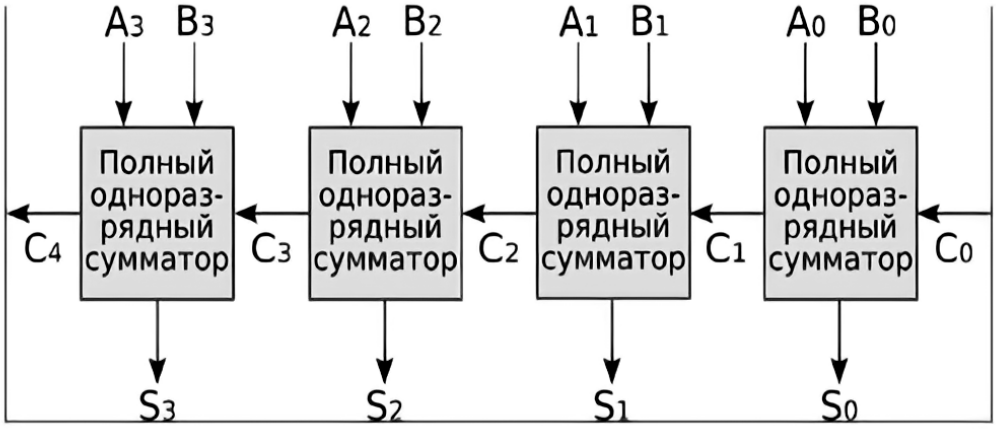


Рисунок 36. Четырёхразрядный сумматор
с последовательным переносом

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 37.

Используя тот же метод, можно объединить четыре четырёхразрядных сумматора в шестнадцатиразрядный (рис. 38). Биты GG и PG, которые схемы ускоренного переноса будут посылать в вышестоящую схему, вычисляются по формулам:

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

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

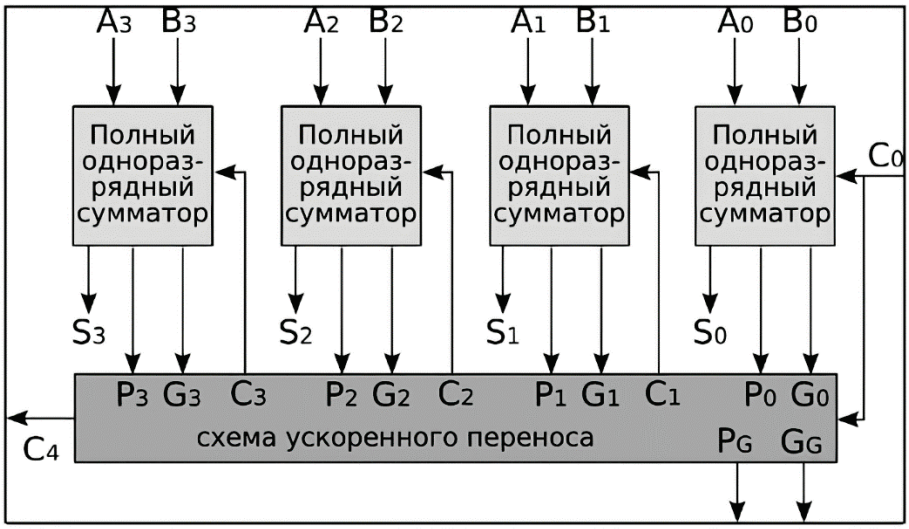


Рисунок 37. Четырёхразрядный сумматор с параллельным переносом

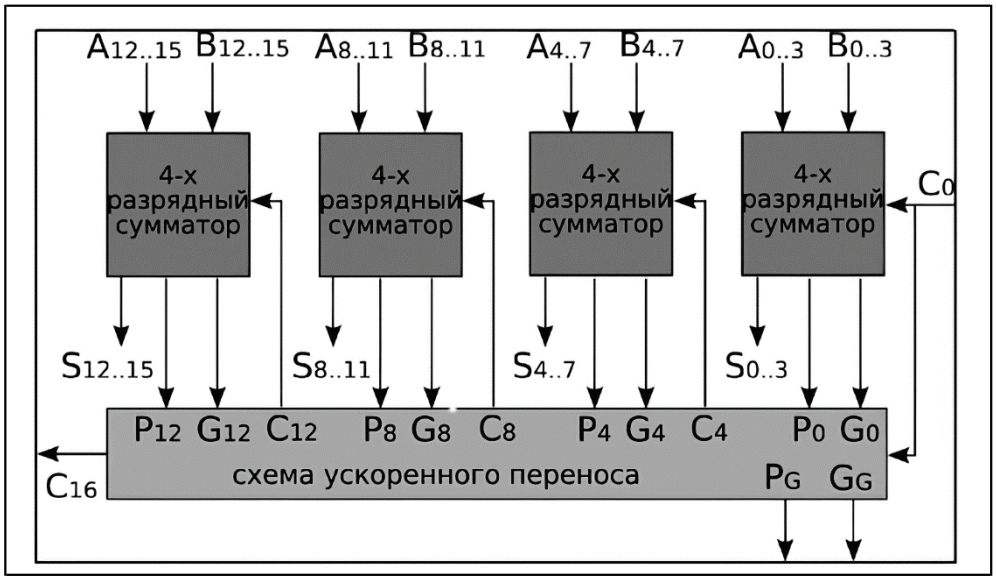


Рисунок 38. Шестнадцатиразрядный сумматор с параллельным переносом

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 39. 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.

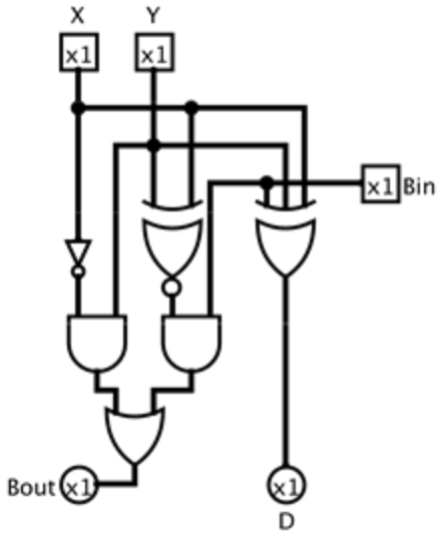


Рисунок 39. Полный одноразрядный вычитатель

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.

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 [5].

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.

Требования к выполнению работы: все задания выполняются в одном файле проекта Logisim; каждое задание должно быть оформлено на отдельной вкладке в виде отдельной схемы с осмысленным названием всех входов и выходов и самой схемы.

После выполнения данной лабораторной работы студенту необходимо сохранить файл и показать преподавателю.

Для защиты данной работы студенту необходимо показать работу устройств в соответствии с таблицей истинности (если таблица не открывается, то воспользоваться калькулятором). Затем подготовить отчет о выполненной работе, где будут прописаны цель работы, ход работы и выводы [5].

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Элементы схемотехники в рамках курса информатики: методические указания к выполнению самостоятельной работы по информатике по теме «Основы схемотехники» для обучающихся по всем программам и форм обучения [Текст] / сост. Ю. С. Бузыкова. – Хабаровск : Изд-во Тихоокеан. гос. ун-та, 2015. – 44 с.

2. Громаков, Е. И. Проектирование автоматизированных систем: Электронный курс лекций [Текст] / Е. И. Громаков. - Томск: Томский политехнический университет, 2009. − 134 с.

3. Logisim графический инструмент для разработки и моделирования логических схем: [сайт]. – URL: http://www.cburch.com/logisim/ (дата обращения: 21.05.2022). – Текст: электронный.

4. Пособие начинающего // Logisim графический инструмент для разработки и моделирования логических схем : [сайт]. – URL: http://www.cburch.com/logisim/docs/2.7/ru/html/guide/tutorial/index.html (дата обращения: 25.05.2022). – Текст: электронный.

5. Попов, Д. И. Организация ЭВМ. Лабораторные работы в программе Logisim [Текст] / Д. И. Попов, И. П. Лилов. – М: МГУП, 2011. – 105 с.

Учебное издание

Сидельников Владимир Иванович
Слюта Марина Олеговна

СИСТЕМЫ АВТОМАТИЧЕСКОЙ ЗАЩИТЫ

Учебно-методическое пособие

Редактор и корректор М. Д. Баранова
Техн. редактор Д. А. Романова Темплан 2021 г., поз. 5258

Подписано к печати 28.02.2023 Формат 60x84/16. Бумага тип № 1.
Печать офсетная. Печ. л. 2,9. Уч.-изд. л. 2,9.
Тираж 30 экз. Изд. № 5258. Цена «С». Заказ №

Ризограф Высшей школы технологии и энергетики СПбГУПТД,
198095, СПб., ул. Ивана Черных, 4