Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №1

# Logisim

Выполнил: Студент: гр. 053506

Ермолович Дмитрий Сергеевич

Руководитель: ст. преподаватель Шиманский В.В.

Минск 2022

СОДЕРЖАНИЕ

1. Введение
2. Постановка задачи
3. Выводы
4. Литература

**Введение**

Цели данной работы:

1. Научиться проектировать и выполнять отладку простых цифровых логических цепей в Logisim.
2. Проектировать конечные автоматы (FSM) и реализовать их как цифровую логическую схему.
3. Приобрести дополнительный опыт проектирования и отладки схем с комбинационной логикой и элементами памяти.

**Постановка задачи**

Как программы на языке С могут содержать вспомогательные функции, так и цепь может содержать вспомогательные подсхемы. В этой части лабораторной работы мы создадим несколько подсхем, чтобы продемонстрировать их использование.

Примечание: Logisim Evolution обычно не разрешает имена с пробелами или символами, имена, начинающиеся с цифр, или имена, конфликтующие с ключевыми словами (например, NAND).

### § **Ход работы**

Выполните следующие шаги. Не забывайте как можно чаще сохранять свою работу**,** не перемещайтеине редактируйте предоставленные контакты входа/выхода**.**

1. Откройте схему задания 1 (File -> Open -> lab01/ex1.circ)
2. Откройте образец подсхем AND2, дважды щелкнув AND2 в селекторе схем слева.

Обратите внимание на 2 в конце; поскольку существует компонент под названием AND, и мы не можем назвать его AND. Мы создали демонстрационную схему для вашего ознакомления. Она имеет 2 1-битных входных контакта, A и B, и посылает результат A & B на выходной контакт RESULT. Она должна выглядеть очень похоже на схему, которую вы только что сделали.

1. Откройте подсхему NAND2. А теперь время сделать вашу собственную цепь! Заполните эту схему, не используя встроенный вентиль NAND из библиотеки вентилей слева (т.е. используйте только вентили AND, OR и NOT; они доступны в виде маленьких иконок на панели инструментов в верхней части окна или в библиотеке вентилей в селекторе схем). Когда вы закончили с этим шагом, аналогично заполните NOR2, XOR2, MUX2 (2 к 1 MUX) и MUX4 (4 к 1 MUX).
   * Пожалуйста, не изменяйте названия подсхем и не создавайте новые, иначе ваша цепь может работать неправильно.
   * Не используйте никаких других встроенных вентилей, кроме AND, OR или NOT. Однако, как только вы создадите подсхему, вы можете (и это приветствуется) использовать её для создания других подсхем. Вы можете сделать это, единожды щелкнув по подсхеме в селекторе схем, а затем разместить ее так же, как вы это делали с вентилями AND/NOT/OR.
   * Это поможет составить таблицу истинности для каждой схемы. Возможно, вам также будет полезно просмотреть слайды лекции о том, как построить эти вентили.
   * Для MUX 4 к 1, SEL0 и SEL1 соответствуют 0-му и 1-му битам 2-битного селектора, соответственно. Следите за тем, чтобы не перепутать их!

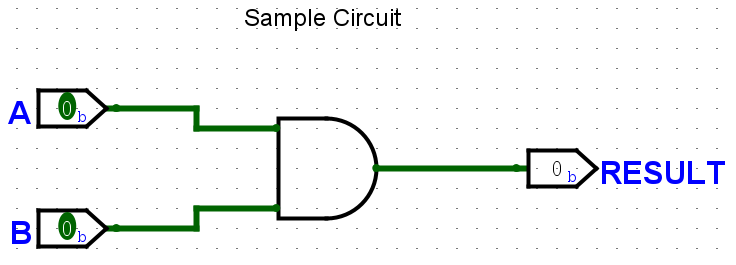


Рисунок 1.1. Схема AND2

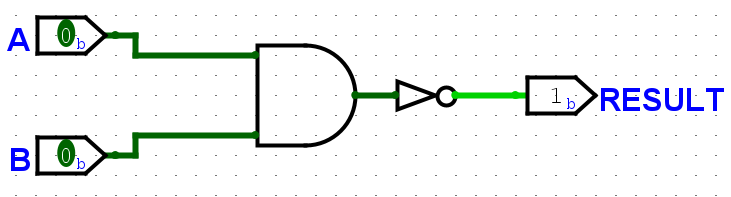


Рисунок 1.2. Схема NAND2

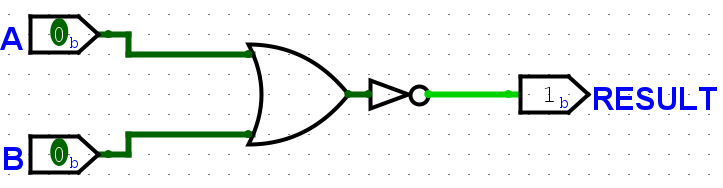


Рисунок 1.3. Схема NOR2

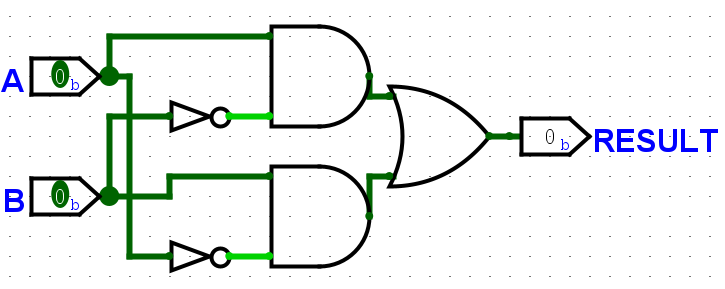


Рисунок 1.4. XOR2

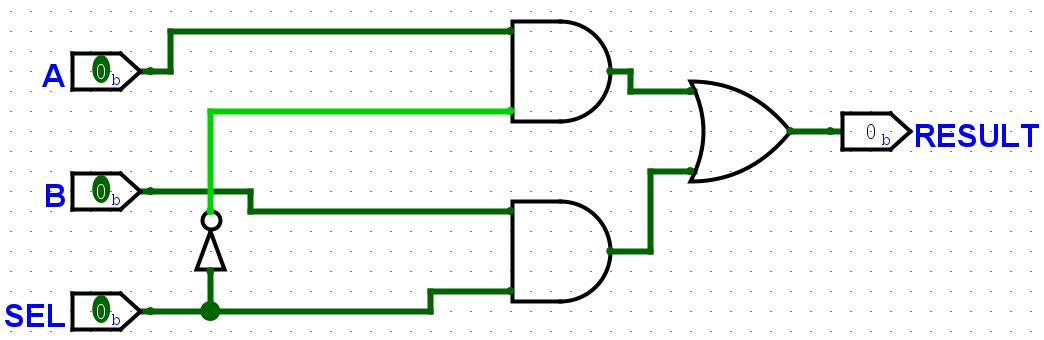


Рисунок 1.5. MUX2

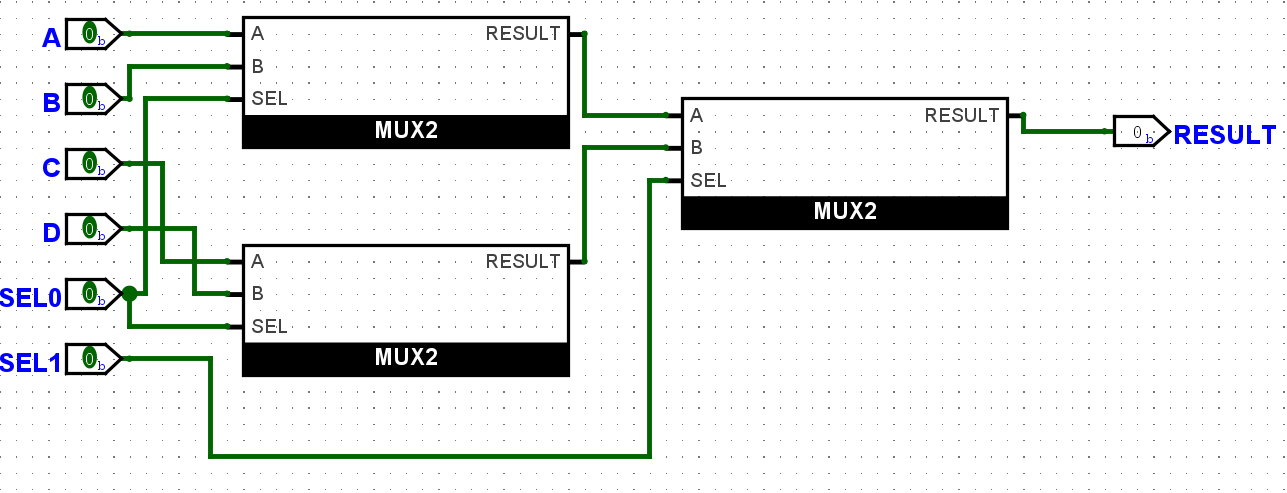


Рисунок 1.6. MUX4

## **Задание 2: Хранение состояния.**

В этом упражнении мы немного забежим вперед и познакомимся с регистровой логикой. Давайте реализуем цепь, которая постоянно увеличивает значение. Разница между этой цепью и цепями, которые вы построили до этого, заключается в том, что она будет хранить это значение как состояние в регистре.

### **Ход работы**

Выполните следующие шаги. Не забывайте как можно чаще сохранять свою работу, не перемещайте и не редактируйте предоставленные контакты входа/выхода.

1. Откройте схему Задания 2 (File -> Open -> lab01/ex2.circ) и перейдите к пустой цепи AddMachine.
2. Выберите подсхему Adder (сумматор) из библиотеки Arithmetic (селектор схем на левой стороне) и поместите сумматор в подсхему AddMachine.
3. Выберите Register из библиотеки Memory и поместите один регистр в вашу подсхему. Ниже приведено изображение, иллюстрирующее части регистра.
4. Подключите входной контакт clk к тактовому контакту вашего регистра. В большинстве ситуаций лучше, чтобы все компоненты в схеме использовали один и тот же тактовый сигнал, чтобы все было синхронизировано. В данном случае тестовое окружение использует тактовый сигнал для своего регистра, поэтому оно передает его через вывод clk для регистров вашей схемы. В будущем, если вы работаете над схемой, у которой нет существующего тактового сигнала, вы можете создать свой собственный, используя новый Clock из библиотеки Wiring.
5. Подключите выход сумматора ко входу регистра, а выход регистра к входу сумматора.
   * При попытке подключения компонентов вы можете получить ошибку "Incompatible widths" (Несовместимые ширины). Это означает, что ваш провод пытается соединить два контакта с разной битовой шириной. Если вы щелкните на компонент с помощью инструмента Selection (значок указателя мыши на панели инструментов в верхней части окна), вы заметите, что в левом нижнем поле окна есть свойство Data Bits. Это значение определяет количество бит на входе и выходе компонента. Убедитесь, что и сумматор, и регистр имеют ширину бита данных 8, после чего ошибка "Incompatible widths" должна быть устранена.
6. Подключите 8-битную константу со значением 1 ко второму входу сумматора. Элемент цепи Constant можно найти в библиотеке Wiring. Чтобы изменить его значение на 1, просто введите 1 в свойстве Value и нажмите Enter. Теперь вы должны увидеть значение 0x1 (Logisim автоматически преобразует введенное десятичное значение в шестнадцатеричное).
7. Подключите два выходных контакта к вашей схеме, чтобы вы могли контролировать то, что выходит из сумматора и регистра. Выход сумматора должен быть подключен к ADD\_OUT, а выход регистра - к REG\_OUT. Таким образом, в итоге ваша схема должна выглядеть следующим образом:
8. Теперь откройте тестовую схему для этого упражнения (lab01/tests/ex2-test.circ). В левом верхнем углу есть небольшая цепь (как ваша AddMachine), отслеживающая текущий цикл. Ниже вы должны увидеть цепь AddMachine, подключенную к тактовому сигналу и некоторым выходным контактам.
9. Запустите импульс на вашу цепь единожды, перейдя в Simulate -> Tick Half Cycle (Command/Control + T). Выходы из вашей AddMachine должны увеличиться! Кроме того, тактовый сигнал теперь должен отображаться ярко-зеленым.

Если вы запустите `Tick Half Cycle` ещё раз, вы увидите, что тактовый сигнал снова отображается тёмно-зелёным, но числа не изменяются. Так происходит, потому что регистры по умолчанию работают по принципу "срабатывания по нарастающей" (т.е. срабатывают при переходе от низкого сигнала к высокому)

Если вы не заинтересованы в наблюдении изменений только для половины циклов, вы также можете использовать `Simulate -> Tick Full Cycle` (`F2`) для запуска тактовых импульсов для полного цикла.

1. Запускать импульсы на цепь вручную может быть довольно утомительно. Хорошие новости: Logisim может делать это за вас! Перейдите в Simulate -> Ticks Enabled/Enable Clock Ticks (Command/Control + K). Теперь через вашу цепь проходят импульсы автоматически!

Если вы хотите, чтобы ваша цепь работала быстрее, вы можете изменить частоту тактов в `Simulate -> Tick Frequency`. Чтобы остановить, снова нажмите `Ticks Enabled//Enable Clock Ticks` (`Command/Control + K`). Если вы хотите сбросить цепь, `Simulate -> Reset Simulation` (`Command/Control + R`).

1. Когда вы закончите, попробуйте запустить предоставленные тесты [используя](https://github.com/PechenkoVkysnoe/AVClabs/tree/main/lab01#§-тестирование) pyhton3 test.py.

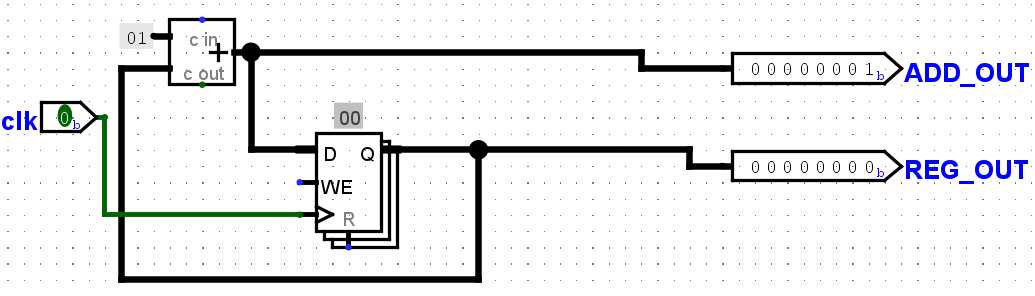


Рисунок 2.1. AddMachine

## **Задание 3: Практика с разветвителями**

Мы собираемся построить цепь, которая манипулирует 8-битным числом.

### **Ход работы**

Выполните следующие шаги. Не забывайте как можно чаще сохранять свою работу, не перемещайте и не редактируйте предоставленные контакты входа/выхода.

1. Откройте ex3.circ и выберите пустую схему SplitThing.
2. Перейдите в библиотеку Wiring и выберите схему Splitter. Эта схема возьмет провод и разделит его на набор проводов меньшей ширины. И так же наоборот, она может взять много наборов проводов и объединить их в один.
3. Измените свойство Bit Width In (ширина шины) на 8, а свойство Fan Out (количество ветвей) на 3. Теперь выберите, какие биты посылать в какую часть вашей ветви. Наименее значимый бит - это бит 0, а наиболее значимый - бит 7. Бит 0 должен выходить на плечо ветви 0, биты 1, 2, 3, 4, 5 и 6 должны выходить на плечо ветви 1, а бит 7 должен выходить на плечо ветви 2.
   * Опция None означает, что выбранный бит не будет выходить ни на одно из плеч ветви.
   * Если вы хотите изменить сразу несколько последовательных битов, щелкните на имя первого из них (не кол-во плеч), нажмите и удерживайте Shift, а затем щелкните на имя последнего. Все строки между ними должны быть выделены. Теперь измените количество плеч одного из битов, и все биты будут обновлены одновременно.
4. Подключите INPUT0 к разветвителю. Подключите 2-входной вентиль AND к плечам ветви 0 и 2 и направьте выход вентиля AND на OUTPUT0.
5. Теперь, интерпретируя ввод как число с методом "знак и величина", разместите логические вентили и другие цепи так, чтобы OUTPUT1 получил отрицательное значение ввода в виде метода "знак и величина". ["Знак и величина"](https://en.wikipedia.org/wiki/Signed_number_representations#Signed_magnitude_representation) (англ. Sign and magnitude) - это альтернативный способ представления знаковых значений такой же, как и дополнительный код (англ. Two's complement), но проще! Для комбинационной логики требуется очень мало вентилей.
6. Нам потребуется другой разветвитель, чтобы рекомбинировать ветви в одну 8-битную шину. Разместите другой разветвитель с соответствующими свойствами (Bit Width In: 8, Fan Out: 3, верные ширины ветвей). "Поиграйте" со свойствами Facing и Appearance, чтобы сделать вашу окончательную схему как можно более опрятней на вид. На данном этапе OUTPUT1 должен быть отрицанием ввода (если интерпретировать INPUT0 И OUTPUT1 как значения метода "знак и величина").
7. Если вы закончили, то попробуйте запустить предоставленные тесты.

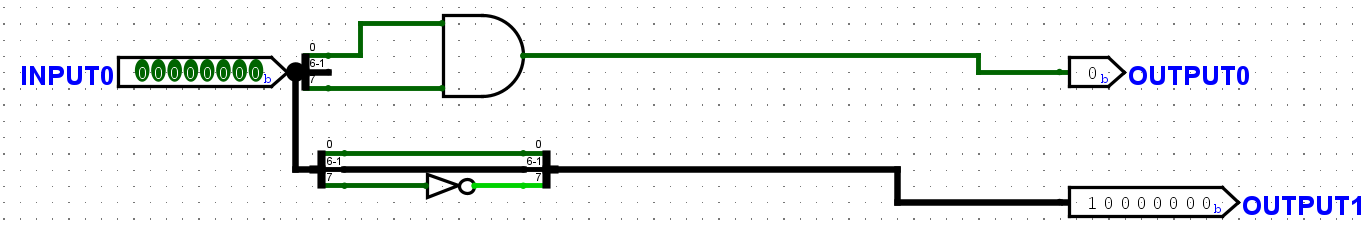
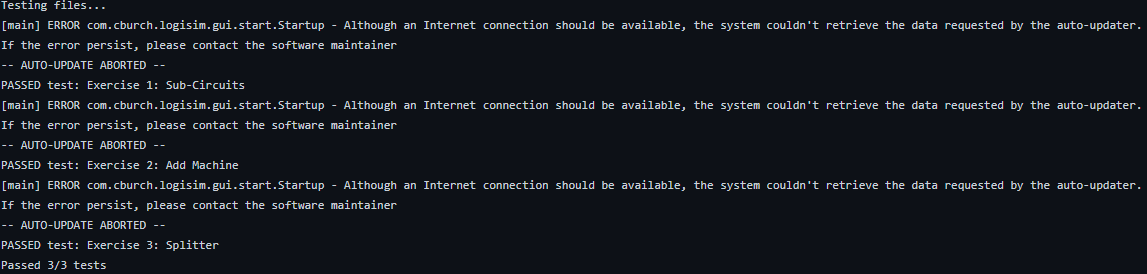


Рисунок 3.1. SplitThing

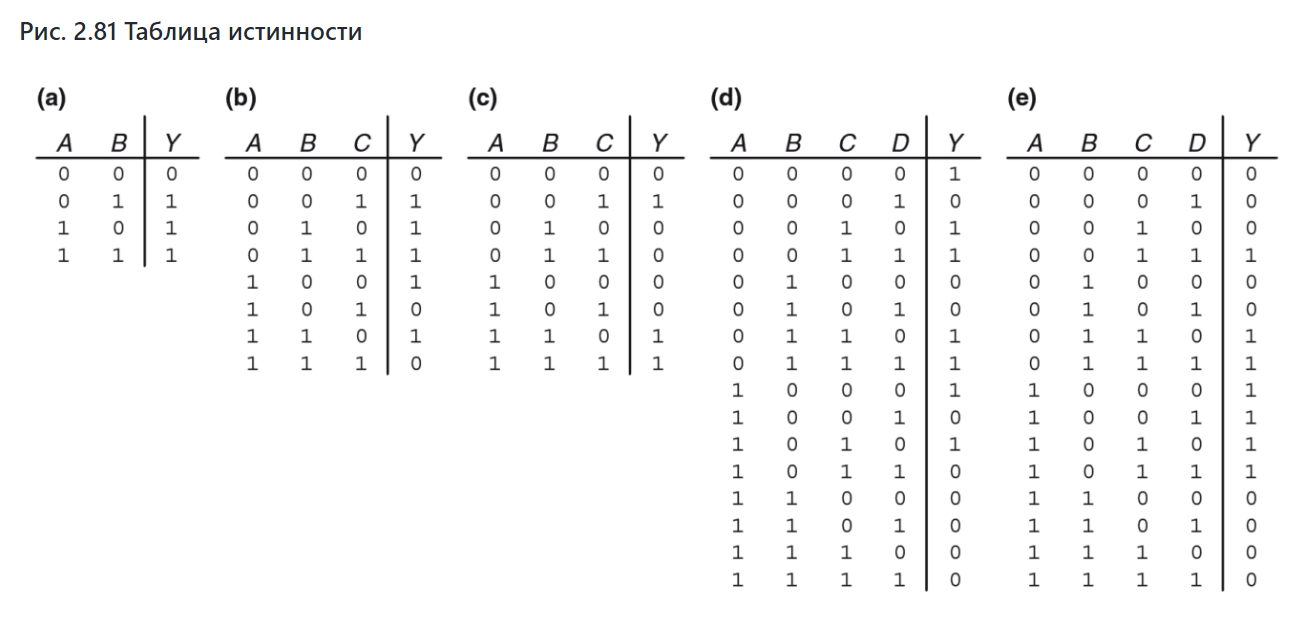
Используем тесты для проверки наших схем:



Все тесты прошли, следовательно задание выполнено правильно.

## **Задание 4: Реализация выражений булевой алгебры.**

### **Рис. 2.81 Таблица истинности**



### **Вариант 8.**

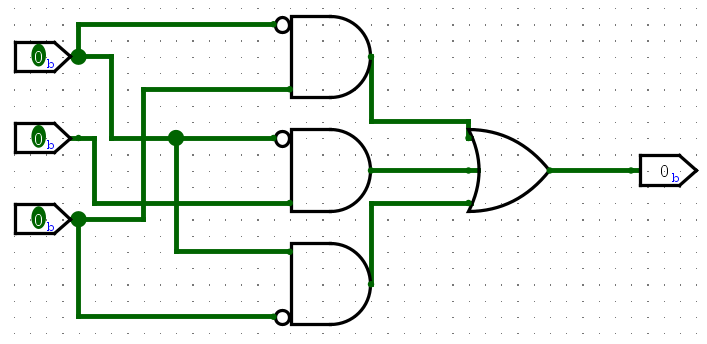
1. Запишите логичесĸое выражение в совершенной дизъюнĸтивной нормальной форме для таблицы истинности (b), приведенной на рис. 2.81.

**Y=!a!bc+!ab!c+!abc+a!b!c+ab!c**

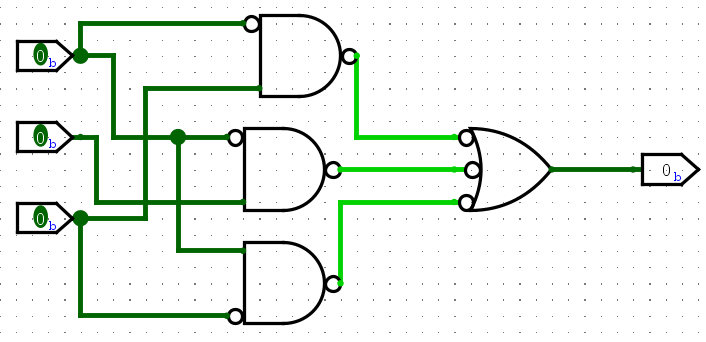
1. Минимизируйте полученное логичесĸое выражение.

**Y=!ac+!ab+a!c**

1. Составьте ĸомбинационную схему, реализующую полученное выражение.

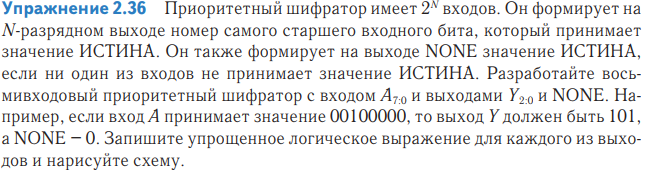


1. Повторите пунĸт 3, используя тольĸо элементы НЕ, И-НЕ и ИЛИ.



## **Задание 5, Вариант 8: Реализация сложных комбинационных схем**.

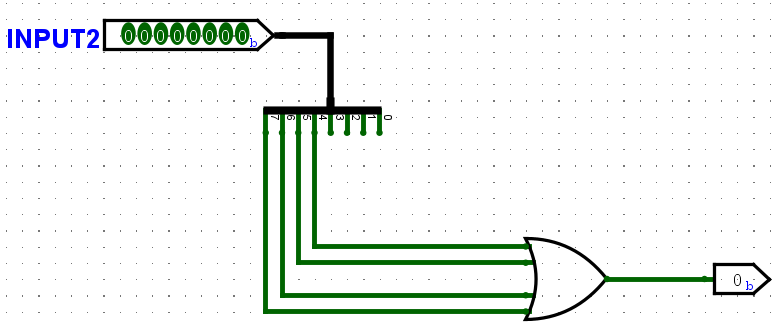
Выполните Упражнение 2.36 из ĸниги Харрис и Харрис ЦИФРОВАЯ СХЕМОТЕХНИКА И АРХИТЕКТУРА КОМПЬЮТЕРА: RISC-V



Составим таблицу истинности схемы приоритета. Затем запишем каждый выход в дизъюнктивной форме и упростим уравнения, используя булевую алгебру.

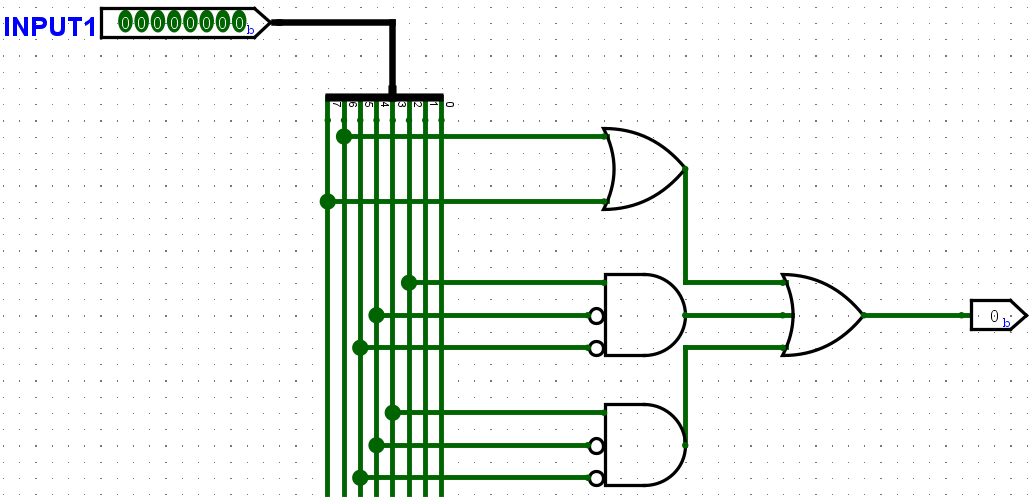
1. Упрощенное логическое выражение для Y2 записывается следующим образом.

**Y2=a4+a5+a6+a7**

****

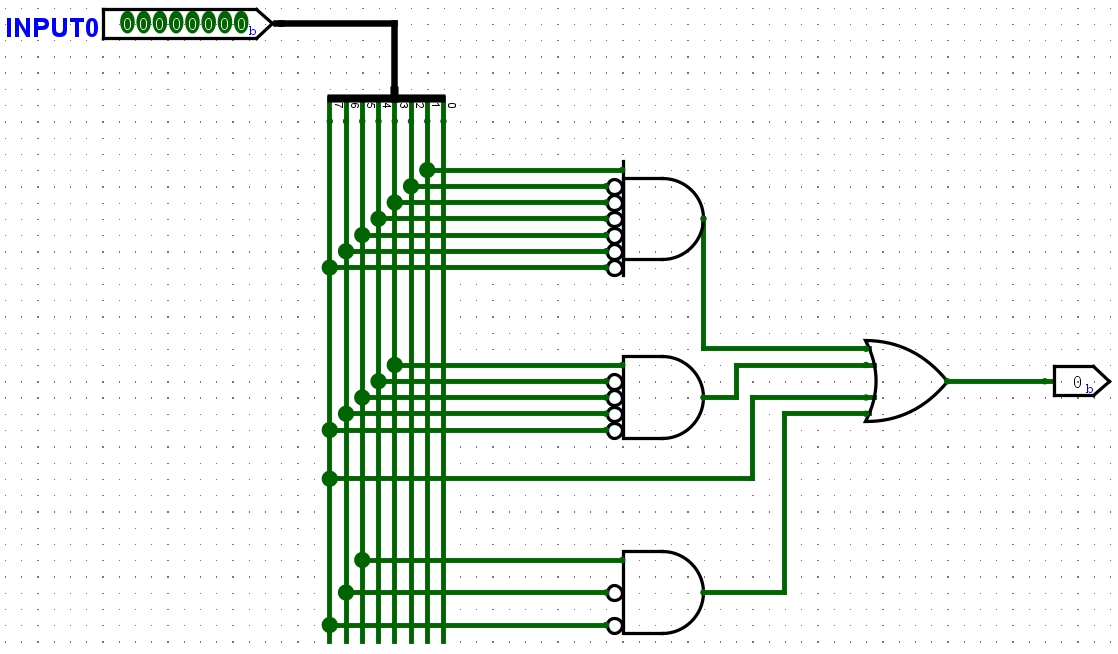
1. Упрощенное логическое выражение для Y1 записывается следующим образом.

**Y1=a6+a7+a2!a4!a5+a3!a4!a5**

****

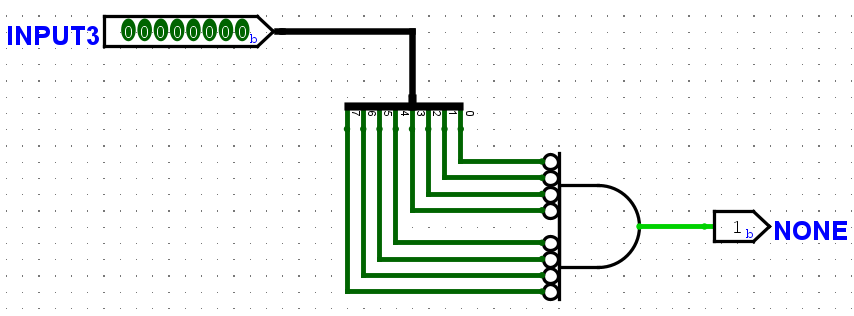
1. Упрощенное логическое выражение для Y0 записывается следующим образом.

**Y0=!a7!a6!a5!a4!a3!a2a1+!a7!a6!a5!a4a3+!a7!a6a5+a7**

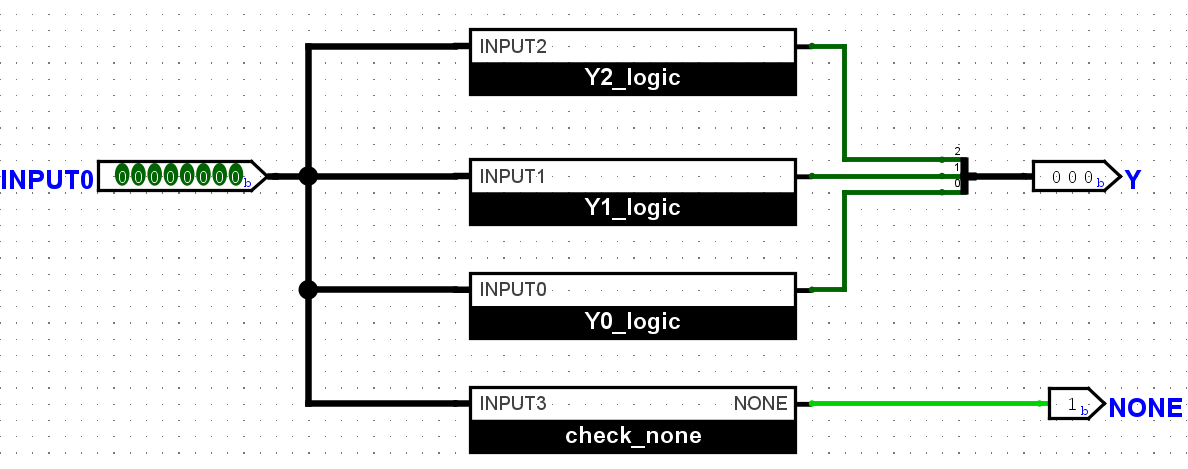
****

1. Упрощенное логическое выражение для NONE записывается следующим образом.

**NONE=!a7!a6!a5!a4!a3!a2!a1!a0**

****

Финальный вид схемы:



**Выводы**

На практике я научиться проектировать и выполнять отладку простых цифровых логических цепей в Logisim, проектировать конечные автоматы (FSM) и реализовать их ĸаĸ цифровую логическую схему. Я получил опыт проектирования и отладки схем комбинационной логики.

**Литература**

Харрис, Дэвид; Харрис, Сара «Цифровая схемотехника и архитектура компьютера. RISC-V» ДМК, 2022.