Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: архитектура вычислительных систем

Отчет по лабораторной работе №1

Logisim

Выполнил: студент: гр. 053506
Слуцкий Никита Сергеевич

Проверил: ст. преподаватель
Шиманский Валерий Владимирович

Минск 2022

СОДЕРЖАНИЕ

Введение

Цели данной работы:

1. Научиться проектировать и выполнять отладку простых цифровых логических цепей в Logisim.
2. Приобрести дополнительный опыт проектирования и отладки схем с комбинационной логикой и элементами памяти.

В ходе выполнения данной лабораторной работы будут получены навыки проектирования простых и нетривиальных логических схем, изучены основы работы с программным продуктом Logisim.

Постановка задачи и результаты выполнения

Задание 1: Подсхемы

Выполните следующие шаги. Не забывайте как можно чаще **сохранять свою работу**, **не перемещайте** и **не 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.

3. 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!

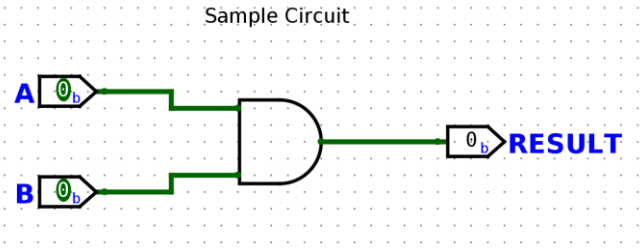

Рисунок 1.1 - реализация схемы AND2

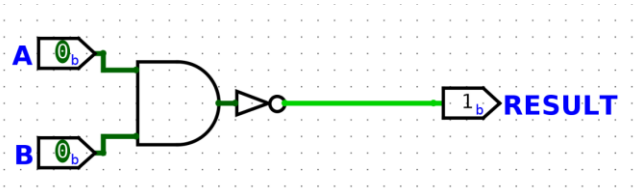

Рисунок 1.2 - реализация схемы NAND2

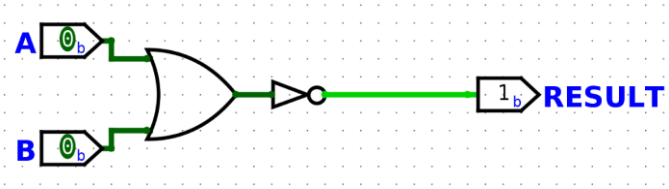

Рисунок 1.3 - реализация схемы NOR2

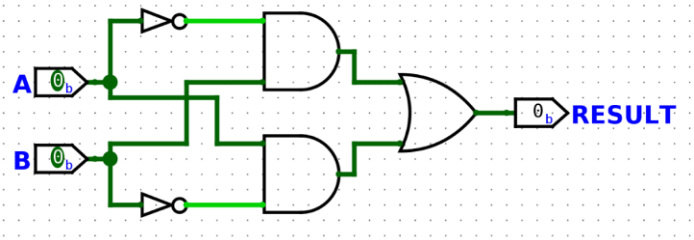

Рисунок 1.4 - реализация схемы XOR2

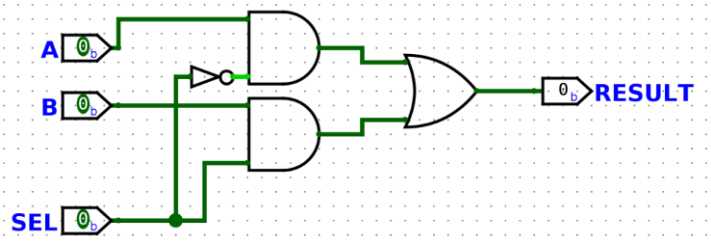

Рисунок 1.5 - реализация схемы MUX2

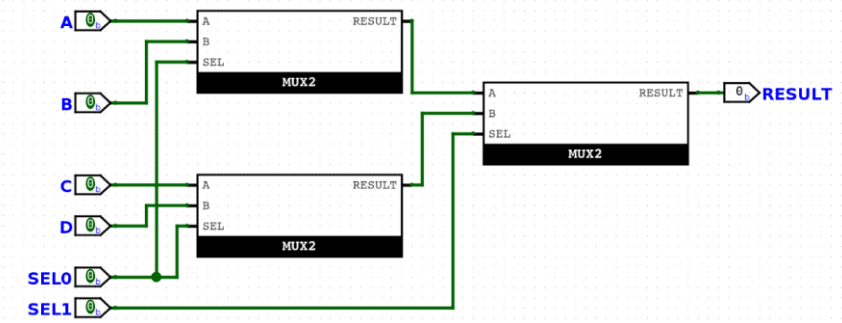

Рисунок 1.6 - реализация схемы MUX4

Четырёхвходовый мультиплексор можно реализовать как на простых компонентах (AND, OR, NOT), так и с использованием трёх уже реализованных двухвходовых мультиплексоров. Лаконичнее выглядит второй вариант. Он представлен выше на рисунке 1.6.

Задание 2: Хранение состояния

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.
    1. 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.
10. 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!
11. 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!

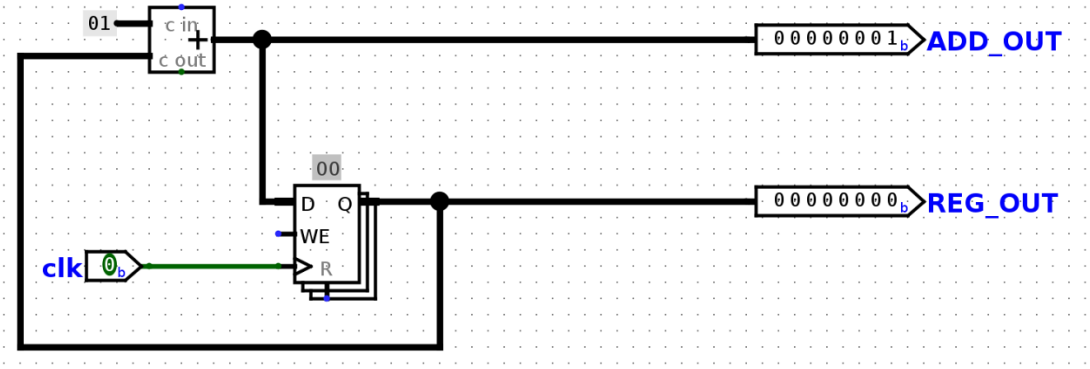

Рисунок 2.1 Реализация схемы AddMachine

Задание 3: Практика с разветвителями

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" (*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.

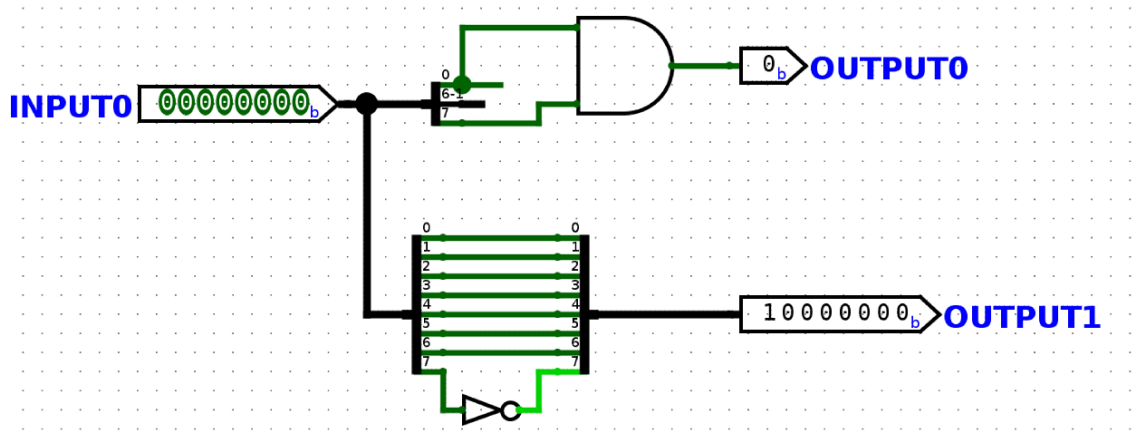

Рисунок 3.1 Реализация схемы для смены в кодировке “Знак и Величина”

Результаты прохождения тестов для первых трёх заданий:

```
Testing files...
PASSED test: Exercise 1: Sub-Circuits
PASSED test: Exercise 2: Add Machine
PASSED test: Exercise 3: Splitter
user@user-of-linux:~/Documents/HomeWork/Architecture
```

Все тесты пройдены — задания выполнены правильно и можно приступать к индивидуальным заданиям 4 и 5.

Задание 4: Реализация выражений булевой алгебры

(a)

A	B	Y
0	0	1
0	1	0
1	0	1
1	1	1

(b)

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(c)

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

(d)

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

(e)

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Вариант 1 (номер в журнале – 21).

1. Запишите логическое выражение в совершенной дизъюнктивной нормальной форме для таблицы истинности (а), приведенной на рис.
2. Минимизируйте полученное логическое выражение.
3. Составьте простую комбинационную схему, реализующую полученное выражение. Под простой схемой подразумевается такая, которая состоит из небольшого количества элементов, но при этом ее разработчик не тратит много времени на проверку каждой из возможных реализаций схемы.
4. Повторите пункт 3, используя только элементы НЕ, И и ИЛИ.

Построю СДНФ (совершенную дизъюнктивную нормальную форму) для заданной таблицей истинности функции **а)**. Для каждого набора переменных, при котором функция равна 1, запишу произведение, причем переменные, которые имеют значение 0 буду брать с отрицанием.

(¬A ∧ ¬B) ∨ (A ∧ ¬B) ∨ (A ∧ B)

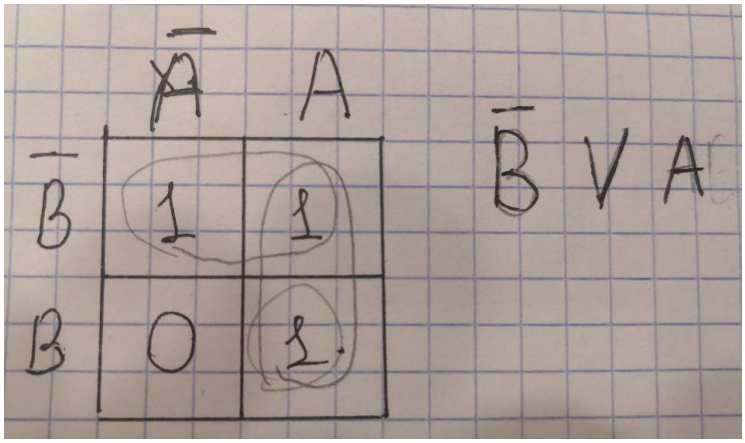

Рисунок 4.1 Карта Карно для минимизации

Итого минимальная ДНФ имеет вид **A ∨ ¬B**

Рисунок 4.2 Соответствующая МДНФ логическая схема

Задание 5: Реализация сложных комбинационных схем

Вариант 1

Требуется реализовать нетривиальный блок комбинационной логики: RotRight, что означает "Rotate Right" (Поворот вправо). Идея заключается в том, что RotRight будет "поворачивать" битовый шаблон входа INPUT0 вправо на AMOUNT (Количество) бит. Таким образом, если INPUT0 был бы 0b1000000000000101, а AMOUNT был бы 0b0011 (3 в десятичном формате), вывод блока был бы 0b1011000000000000. Обратите внимание, что крайние 3 бита были повернуты от правого конца значения к левому. Это можно представить как побитовую операцию R = A >> B | A << (16 - B).

- Реализуйте подсхему с названием RotRight со следующими входами:
- INPUT0 (16-бит) 16-битный вход, который необходимо повернуть
- AMOUNT (4-бит) количество оборотов (почему 4 бита?)
- Выполните следующие шаги. Не забывайте как можно чаще **сохранять свою работу**.

- В качестве выхода должен быть INPUT0, повернутый вправо на AMOUNT бит, как описано выше. Вам **НЕ** разрешается использовать в вашем решении шифтеры Logisim, хотя вся остальная комбинационная логика (MUX'ы, константы, вентили, сумматоры и т.д.) разрешена. Особенно полезными могут оказаться встроенные в Logisim MUX'ы (их можно найти в меню Plexers). Ваше решение не должно включать такты или какие-либо элементы с тактовой частотой, например, регистры.

- До того, как вы приступите к подключению, вам следует хорошо подумать о том, как разложить эту задачу на более мелкие задачи, а затем соединить их вместе. Вы можете **использовать подсхемы** при реализации RotRight Если вы этого не сделаете, все может усложниться.

- То, что операция представляется в виде побитовой операции, не означает, что это лучший способ решения задачи. Подумайте о входных битах B и подумайте о том, как можно эффективно использовать разветвители! Можете ли вы что-то сделать с двоичной формой? Вспомните, почему двоичная форма удобна для использования в компьютерах: 1 легко представить как сигнал ON, а 0 - как сигнал OFF. Допустим, мы хотим повернуть 9 раз. 9 - это 1001 в двоичном формате, или 1*8 + 0*4 + 0*2 + 1*1. Можете ли вы использовать это для создания более чистой цепи? Использование предоставленных нами схем rot* - это хорошая идея, которая позволит сохранить чистоту в нашей работе!

В данном задании все операции сведены к более простым. Реализован компонент, осуществляющий побитовый циклический сдвиг (по условию это называется “поворот”) вправо на одну позицию. Далее из двух таких собран компонент для двойного сдвига. Далее – для сдвига на 4 и 8 позиций. Использована подсказка того, что число, на которое осуществляется сдвиг, можно разложить по степеням двойки. С помощью разветвителя берётся значение бита и, если оно равняется 1, то осуществляется соответствующий сдвиг (на 1, 2, 4 или 8 позиций с применением вышеописанных реализованных компонентов). Например, если в числе, которое указывает число для сдвига, на третьем бите стоит 1 (при нумерации с единицы), то есть оно имеет вид x1xx, то исходное число пройдёт через компонент RotRight4. И так далее.

В итоге число проходит через “цепочку” преобразований, где на каждом шаге смотрится: если бит равен 1, то осуществляется соответствующий сдвиг (на 1, 2, 4 или 8 позиций), иначе – последовательность остаётся неизменной. Выборку позволяет реализовать двухвходовый мультиплексор, где вариантами для выбора являются “нетронутая” (пришедшая с прошлого шага) и сдвинутая последовательность, а дополнительным сигналом для выбора – значение текущего рассматриваемого бита. Таким образом итоговая схема смотрится лаконично, незагромождённо и понятно.

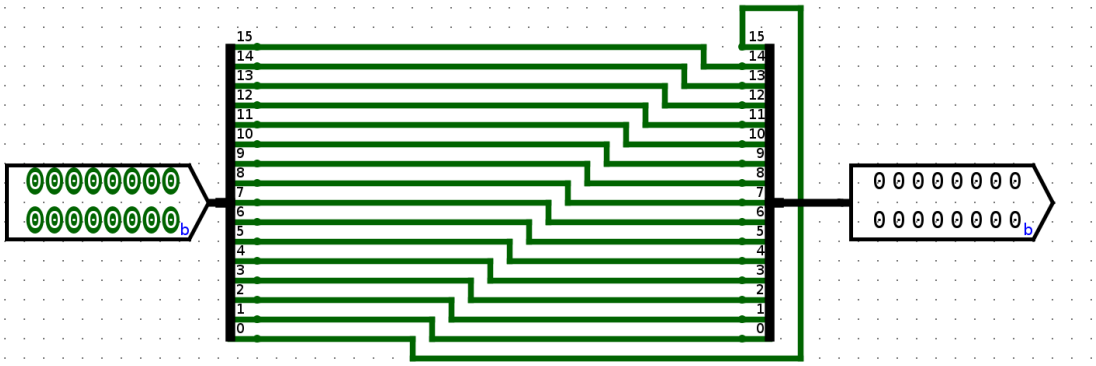

Рисунок 5.1 Реализация циклического сдвига (поворота) вправо на одну позицию

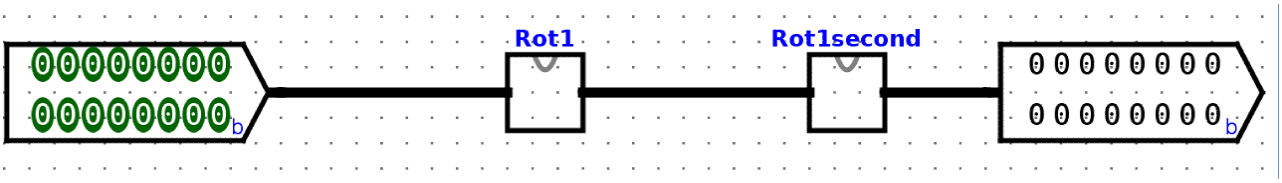

Рисунок 5.2 Реализация циклического сдвига (поворота) вправо на две позиции (которая сводится к применению дважды прошлого компонента с рис. 5.1)

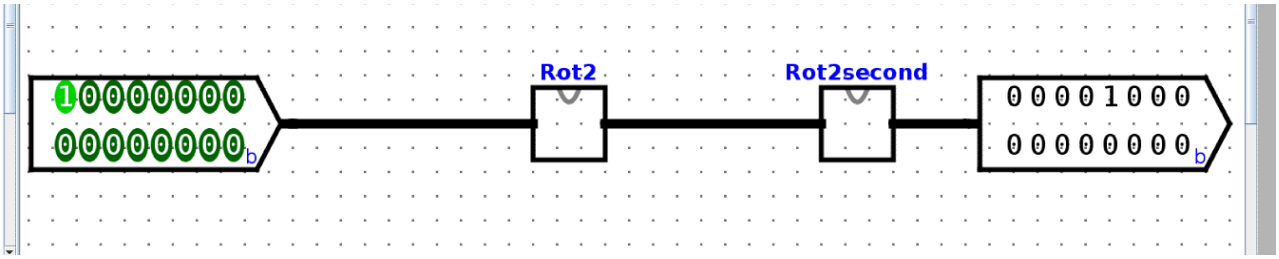

Рисунок 5.3 Реализация циклического сдвига (поворота) вправо на четыре позиции (которая сводится к применению дважды прошлого компонента с рис. 5.2)

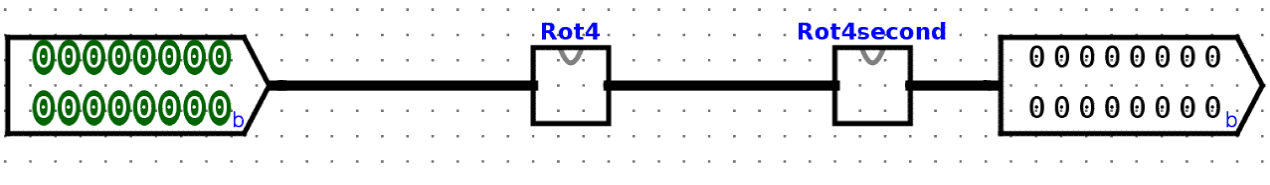

Рисунок 5.4 Реализация циклического сдвига (поворота) вправо на восемь позиций (которая сводится к применению дважды прошлого компонента с рис. 5.3)

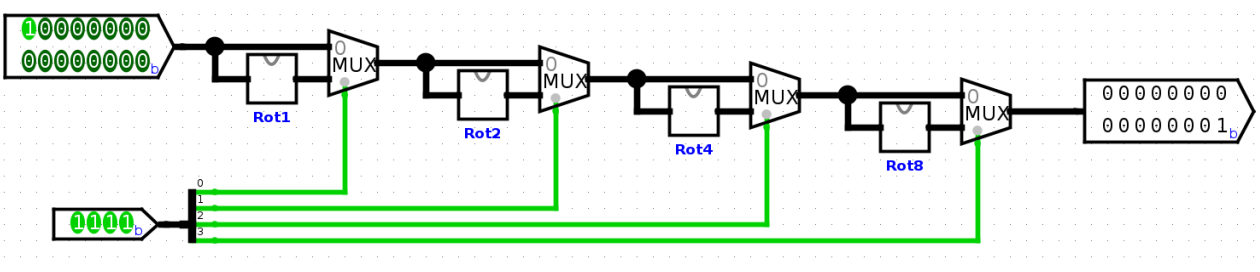

Рисунок 5.5 Основная схема, выполняющая поставленную задачу

Выводы

В результате выполнения лабораторной работы №1 на практике были изучены основы работы с программным продуктом Logisim и применены существующие знания для разработки простых и не очень логических схем. Были составлены схемы логического AND, NAND, XOR, NOR, MUX(2), MUX(4) используя примитивные логические элементы.

Используя более сложные элементы и более сложную логику были составлены следующие схемы AddMachine, SplitThing.

Реализованы выражения булевой алгебры, а также сложные комбинационные схемы.

Работа к сдаче готова.

Литература

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