

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ
НАПРАВЛЕНИЕ СИСТЕМНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО
ОБЕСПЕЧЕНИЯ

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

курса «Функциональная схемотехника»

**по теме: «Введение в проектирование цифровых интегральных
схем»**

Вариант № 6

Выполнил студент:

Тюрин Иван Николаевич

группа: Р33102

Преподаватель:

Кустарев П. В.,

Васильев С.Е.

Санкт-Петербург, 2024 г.

Содержание

| | |
|--|----|
| Лабораторная работа № 1. Введение в проектирование цифро- вых интегральных схем | 2 |
| 1. Цели работы | 2 |
| 2. Задание варианта № 6 | 2 |
| 1. Часть 1 | 2 |
| 2. Часть 2 | 3 |
| 3. Выполнение задания | 4 |
| 1. Часть 1: построение схемы в LTSpice | 4 |
| 2. Часть 2: описание схемы в Vivado | 10 |
| 4. Вывод | 14 |

Лабораторная работа № 1

Введение в проектирование цифровых интегральных схем

1. Цели работы

1. Получить базовые знания о принципах построения цифровых интегральных схем с использованием технологии КМОП.
2. Познакомиться с технологией SPICE-моделирования схем на транзисторах.
3. Получить навыки описания схем базовых операционных элементов (БОЭ) комбинационного типа на вентиляном уровне с использованием языка описания аппаратуры Verilog HDL.

2. Задание варианта № 6

2.1. Часть 1

1. Постройте в LTspice на транзисторах схему вентиля, составляющего основу логического базиса согласно варианту задания.
2. Создайте символ для разработанного вентиля как иерархического элемента.
3. С использованием созданного иерархического элемента постройте схему тестирования вентиля.
4. Проведите моделирование работы схемы и определите задержку распространения сигнала через тестируемый вентиль.
5. Определите максимальную частоту изменения входных сигналов, при которой построенная схема сохраняет работоспособность.

6. Постройте БОЭ на базе созданного вентиля согласно варианту задания.
7. Создайте символ для построенного БОЭ.
8. Проведите моделирование работы схемы и определите задержку распространения сигнала через БОЭ.
9. Определите максимальную частоту изменения входных сигналов, при которой построенная схема сохраняет работоспособность.
10. Составьте отчет по результатам выполнения заданий первой части лабораторной работы.

2.2. Часть 2

1. Опишите на Verilog HDL на вентильном уровне модуль, реализующий функцию БОЭ в указанном логическом базисе согласно варианту задания.
2. Разработайте тестовое окружение для созданного модуля.
3. Проведите моделирование работы схемы.
4. Составьте отчет по результатам выполнения заданий второй части лабораторной работы.

Вариант:

| № варианта | Логический базис | БОЭ |
|------------|------------------|--------------------------------|
| 6 | NAND | Позиционный дешифратор «3 в 8» |

Таблица 1.1: Вариант задания

, , ,

3. Выполнение задания

В соответствии с заданием необходимо было описать схему базового операционного элемента (БОЭ), декодирующего 3 входных в 8 выходных сигналов по принципу двоичной записи числа: выходное значение представляет собой двоичное число равное степени числа 2 с показателем, представленным двоичным числом на входе.

3.1. Часть 1: построение схемы в LTSpice

Для построения требуемого БОЭ в LTSpice нужно построить схему и изобразить символ элемента используемого как логический базис, т.е. функцию NAND, суть которого можно видеть на таблице истинности [1.2](#).

| A | B | NAND |
|---|---|------|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Таблица 1.2: Таблица истинности функции NAND

Логический элемент NAND был сконструирован с помощью схемы представленной на изображении [1.2](#). В элементе используется выделенный входной порт для питания VDD, чтобы обеспечить независимость работы компонента от внутреннего элемента питания: таким образом достигается установка глобального уровня сигнала соответствующего логической единицы, с которым схема «сравнивает» входные сигналы. Так же для элемента NAND был разработан графический символ соответствующий стандарту ANSI, его можно видеть на изображении [1.1](#).

Базисный логический элемент был протестирован с помощью схемы представленной на изображении [1.3](#). При этом, с целью пронаблюдать работу элемента NAND со всеми возможными комбинациями входных значений во время симуляции, в схеме были использованы импульсные элементы питания с различными периодами пульсации. А в качестве нагрузки использовался резистор и конденсатора с характеристиками указанными в задании.

Нагрузка представляет собой сопротивление обладающее как активным, так и реактивным сопротивлением. Именно это сопротивление позволяет получить довольно плавное изменение уровня выходного напряжения на элементе NAND и тем самым достаточно качественно моделирует нагрузку в виде более сложной функциональной схемы (зависимость уровня сигнала от характеристик нагрузки можно будет наблюдать далее).

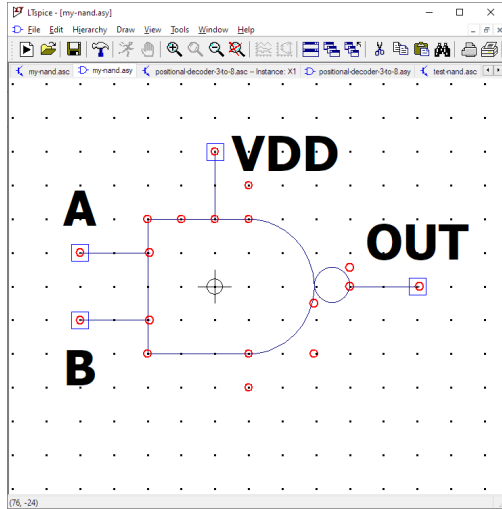


Рис. 1.1: Символ используемый для разработанного вентиля NAND.

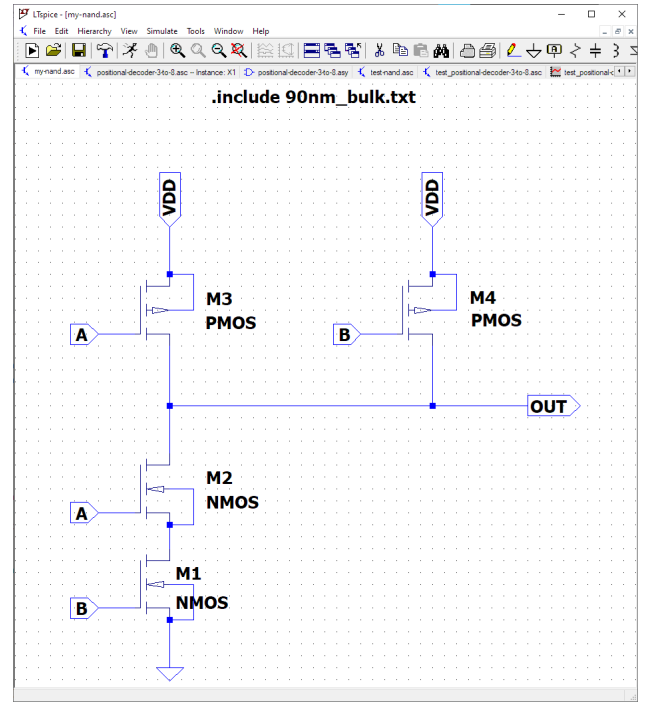


Рис. 1.2: Схема разработанного вентиля NAND.

Результат симуляции работы разработанной тестирующей схемы можно видеть на изображении 1.4 и более подробно с большим масштабом и дополнительными обозначениями на изображении 1.5.

Для элемента NAND вычислим максимальную частоту для корректной его работы. Будем считать, что логический высокий уровень начинается на значении 0,8 (В), а низкий на значении 0,2 (В). Тогда, исходя из результатов моделирования 1.5, длительность фронта сигнала и длительность спада на декодере равны

$$T_{\text{rise}} \approx 0,7 \text{ (нс)} \text{ и}$$

$$T_{\text{fall}} \approx 0,9 \text{ (нс)}$$

соответственно, а длительность задержки тогда

$$T_{\text{delay}} = T_{\text{rise}} + T_{\text{fall}} = 1,6 \text{ (нс)}.$$

Максимально допустимую частоту работы схемы можно определить по формуле

$$F_{\text{max}} = \frac{1}{T_{\text{delay}}} = 0,625 \text{ (ГГц)}.$$

Далее с использованием элемента NAND был построен БОЭ декодер, изображение схемы которого можно видеть на изображении 1.6. Для удобного его проектирования использовались дополнительные порты для передачи

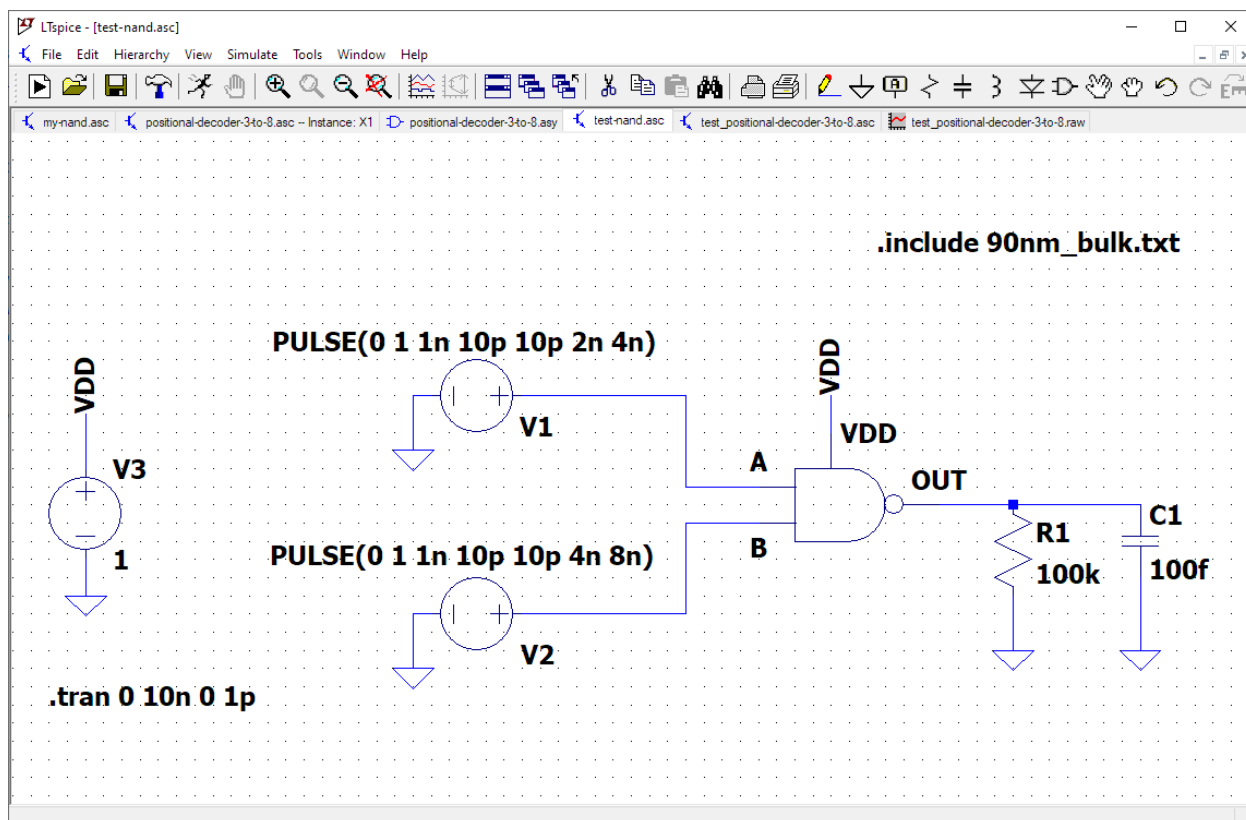


Рис. 1.3: Схема разработанная для тестирования компонента NAND.

инвертированных значений входных сигналов. А для инверсии значения сигнала использовались компоненты NAND, которым на оба входа подаются одинаковые значения — так NAND вполне компактно реализует логическую функцию NOT.

При разработке схемы использовалась идея, заключающаяся в том, что NAND дает на выход значение, которое можно однозначно интерпретировать как комбинацию входных значения только при условии, что оба входа поданы логические единицы: только в этом случае можно использовать значение на выходе для определения состояния на обоих входах.

Также, изначально значение требуемого входного порта, отвечающего за включение элемента (Enable), было ошибочно использовано как вход питания (логический высокий уровень, Vdd) для элементов NAND, что обеспечивает нулевое значение на выходе декодера в случае логического нуля на входе его питания, но в конечном итоге пришли к выводу, что вход Enable должен отдельно контролировать поступление входного сигнала на выход, а для питания NAND элементов в схему был добавлен еще один входной порт, принимающий логический высокий уровень сигнала.

Для декодера был разработан графический символ, в котором 3 входных логических значения A, B, C и входное значение питания E. Внешний вид символа можно видеть на изображении 1.7.

Разработанный декодер был протестирован с использованием отдельной

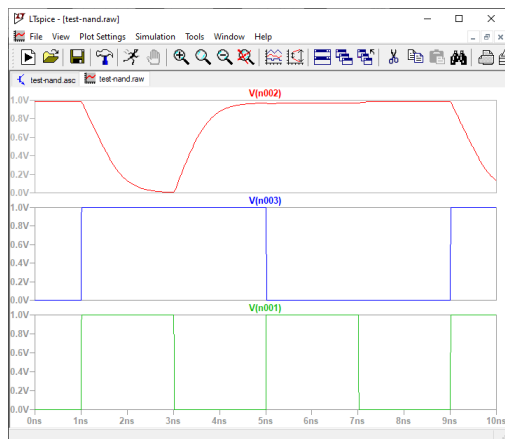


Рис. 1.4: Временная диаграмма симуляции работы тестирующей схемы для элемента NAND.

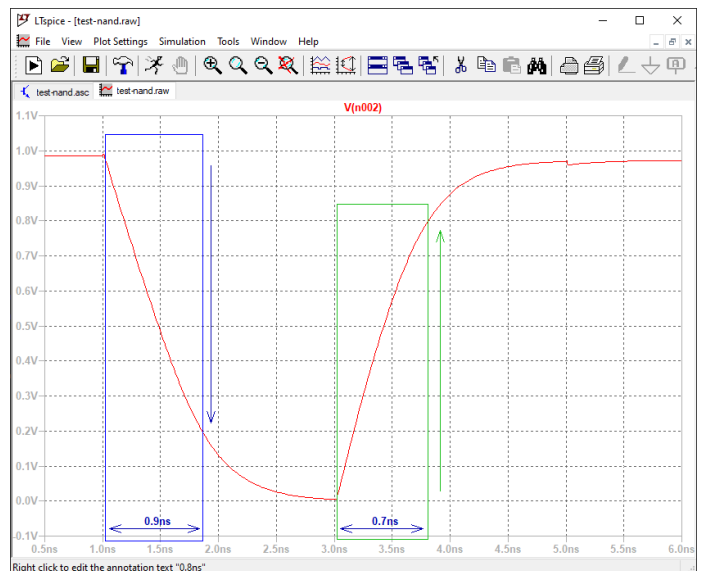


Рис. 1.5: Исследование временной диаграммы работы тестирующей схемы для элемента NAND на большем масштабе.

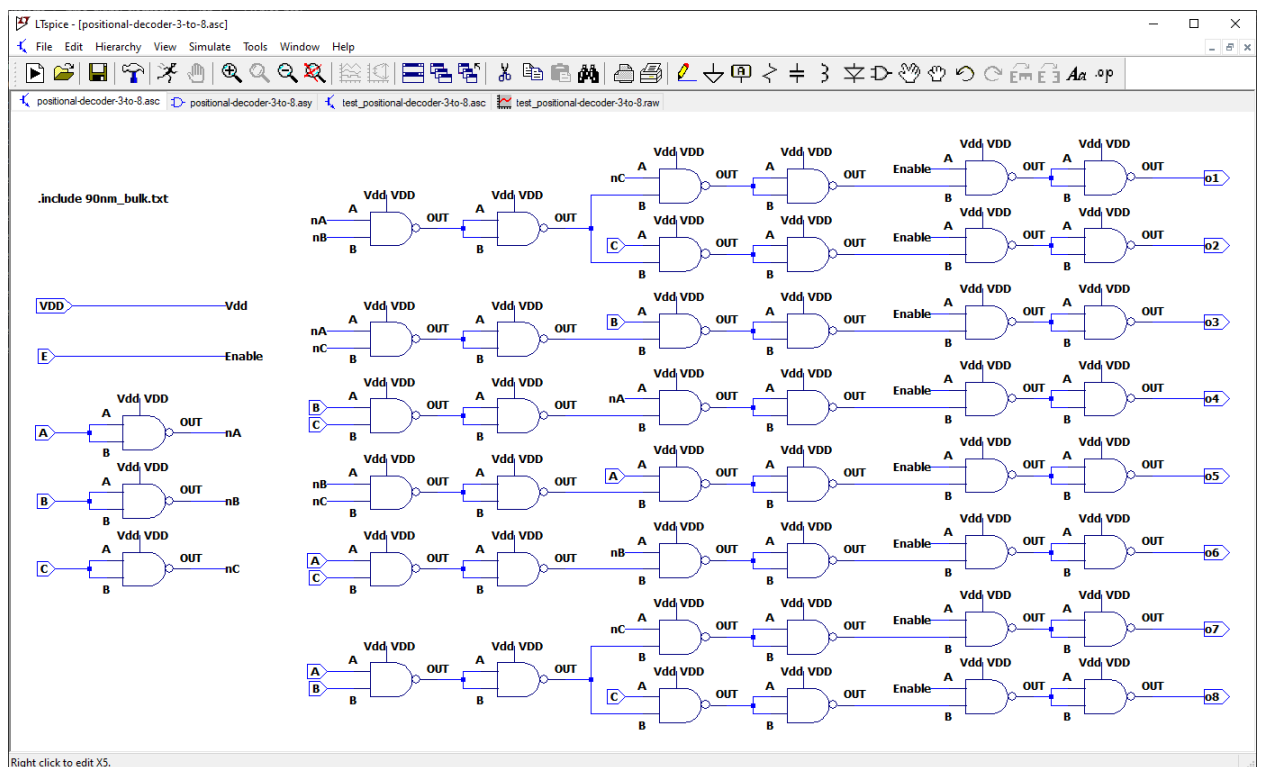


Рис. 1.6: Схема разработанного позиционного декодера с использованием ранее разработанного логического базиса (NAND)

схемы, представленной на изображении 1.8. В ней на вход БОЭ подавались значения с разных импульсных элементов питания, которые обладали кратными периодами пульсации. Более того, элемент питания для входа E (Enable) декодера тоже был импульсным и при том со смещением периода относительно других источников питания, чтобы продемонстриро-

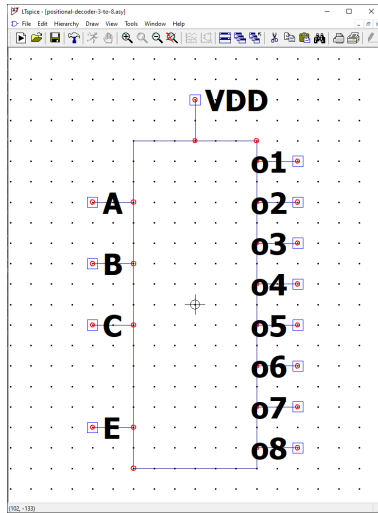


Рис. 1.7: Символ позиционного декодера.

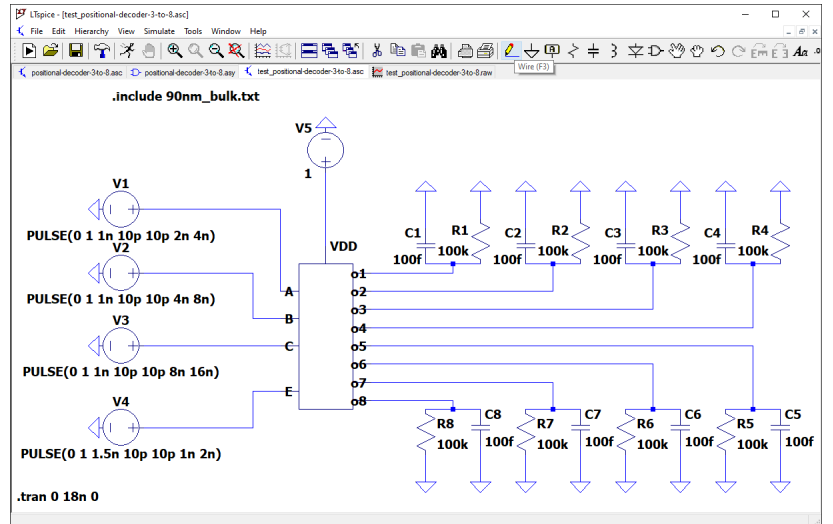


Рис. 1.8: Схема для тестирования функциональности разработанного декодера.

вать зависимость результата работы декодера от этого входного сигнала. Каждый выходной порт декодера был подключен к различному элементу нагрузки, чтобы исключить их замыкания. В качестве нагрузки на компонент используются параллельно подключенные резистор и конденсатор с характеристиками, указанными в условии лабораторной работы.

Во время симуляции работы декодера можно хорошо наблюдать изменение значений на выходах декодера в разных комбинациях входных значений (см. графики на рисунке 1.9). При этом все изменения выходных значений происходят строго в соответствии с входным значением Enable и так же им присущи переходные процессы.

На временной диаграмме 1.10 симуляции можно измерить задержки прохождения сигнала через схему БОЭ. Изменения состояния будем считать, аналогично тому, как это делали для логического элемента, т.е. 0,8 (В) — верхний уровень, 0,2 (в) — нижний. Тогда длительность фронта сигнала на декодере и длительность спада

$$T_{\text{rise}} \approx 0,4 \text{ (нс)} \text{ и}$$

$$T_{\text{fall}} \approx 0,9 \text{ (нс)}$$

соответственно, а длительность задержки равняется

$$T_{\text{delay}} = T_{\text{rise}} + T_{\text{fall}} = 1,3 \text{ (нс)}.$$

Максимально допустимую частоту работы схемы можно определить по формуле

$$F_{\text{max}} = \frac{1}{T_{\text{delay}}} \approx 0,77 \text{ (ГГц)}.$$

Тут же можно заметить, что задержка фронта БОЭ (декодера) отличается от задержки базисного элемента (NAND): 0,4 и 0,7 (нс) соответственно.

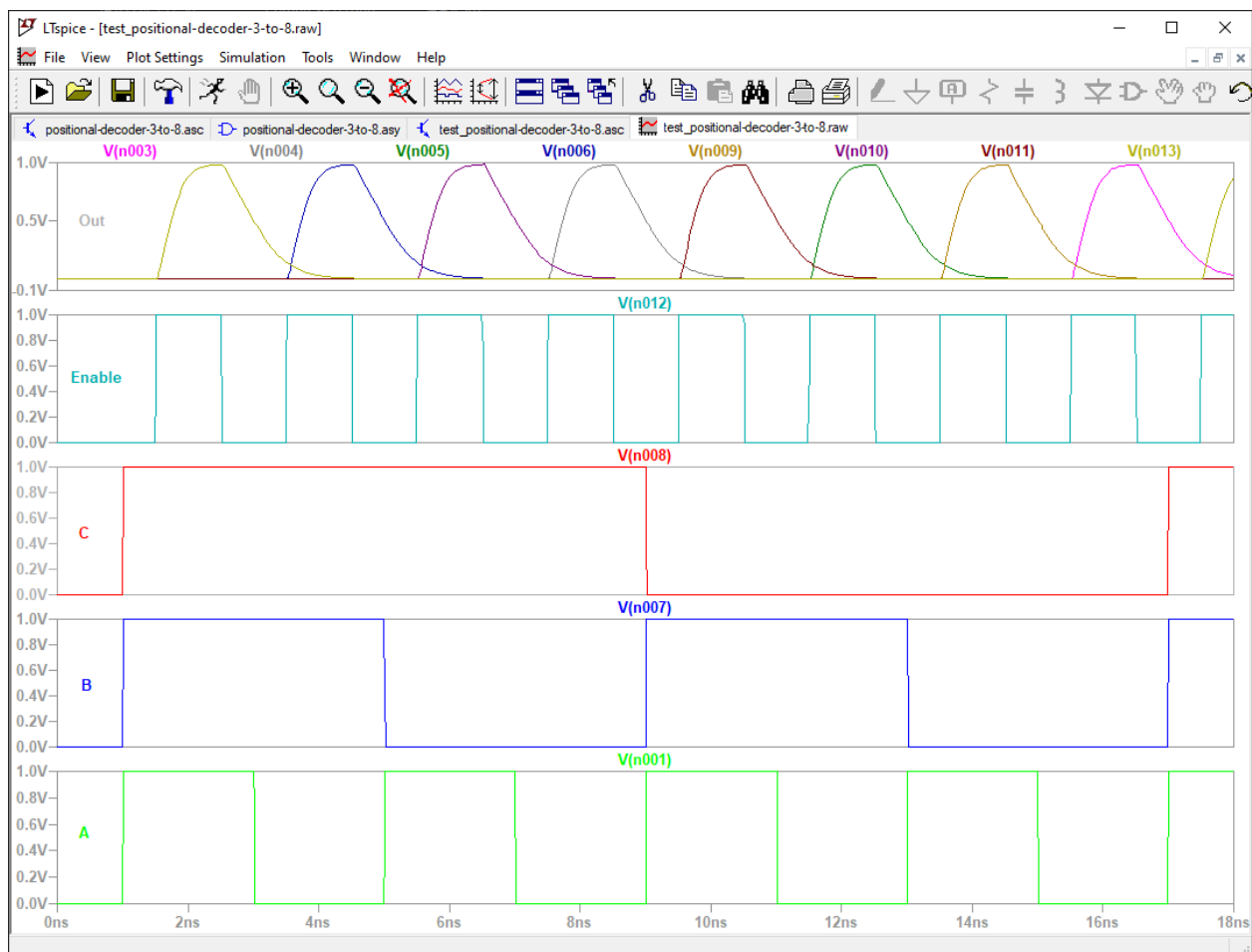


Рис. 1.9: Временная диаграмма результата симуляции входных и выходных значений на разработанном декодере.

Считая, что среда разработки симулирует верно, можно предположить, что уменьшение задержки вызвано именно сложной комбинацией базисных элементов, что вызывает, к примеру, снижение общей электроёмкости функционального компонента; активное сопротивление диодной схемы при сложных соединениях практически не меняется.

Для проверки этой теории был произведен тест с изменением ёмкости конденсатора в нагрузке для конкретного выходного порта. На временной диаграмме симуляции, представленной на изображении 1.11, можно видеть, что при увеличении ёмкости конденсатора в нагрузке, задержка фронта увеличивается приблизительно до ожидаемого значения 0,7 (нс), но также меняется форма сигнала. Похоже, что при достижении нужного соотношения между ёмкостями функциональных элементов задержки принимают определенные значения.

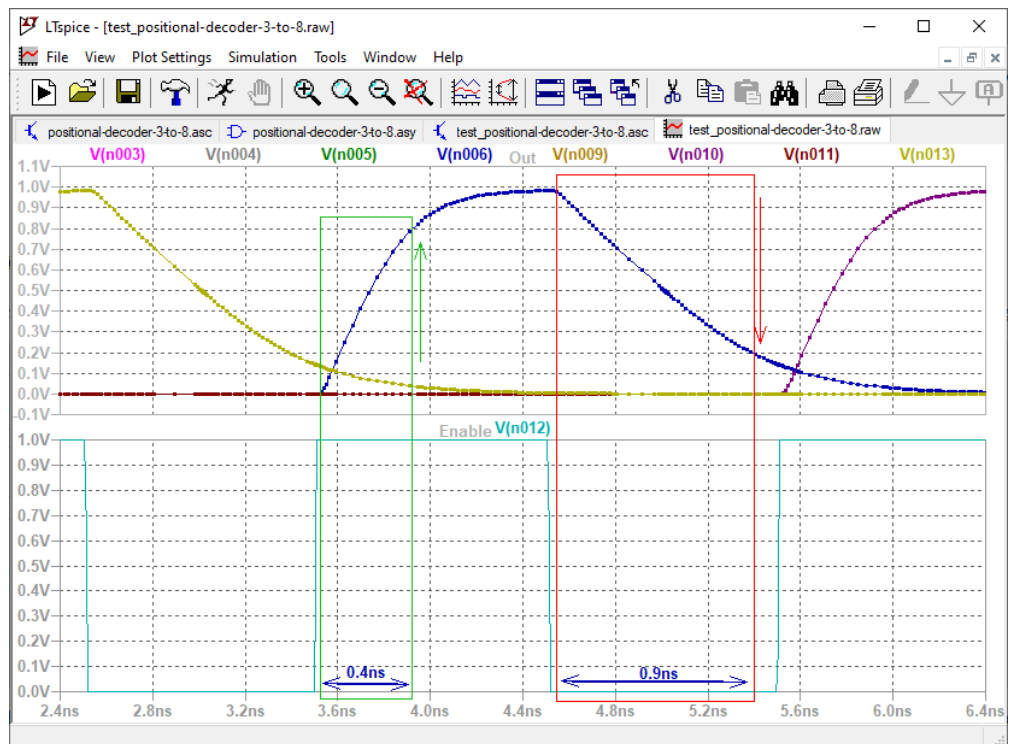


Рис. 1.10: Исследование временной диаграммы симуляции работы разработанного декодера на большем масштабе.

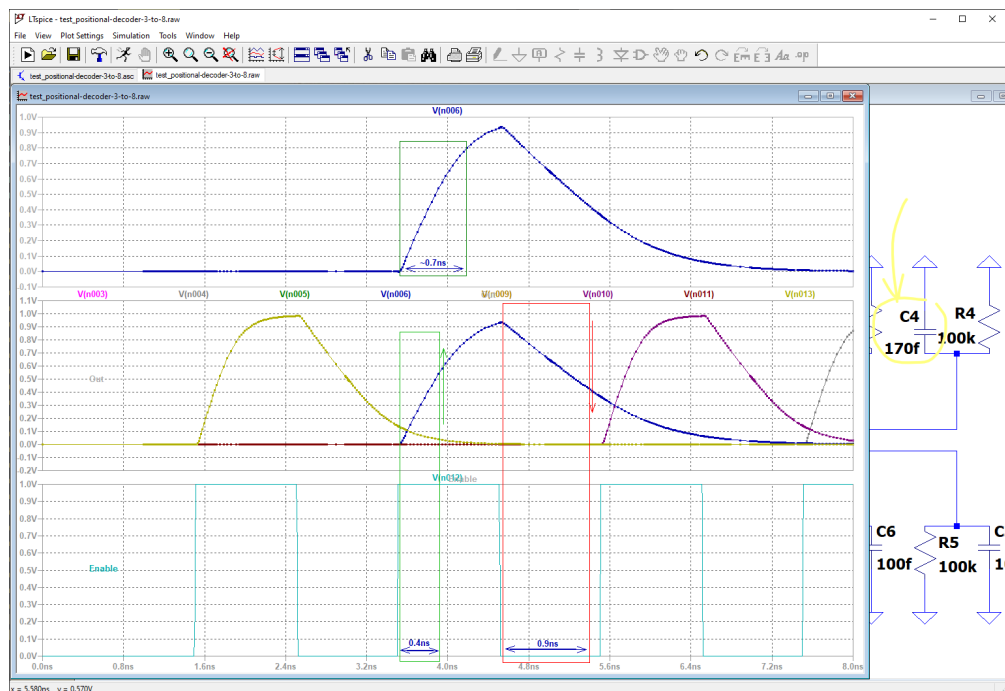


Рис. 1.11: Исследование временной диаграммы симуляции с измененными характеристиками нагрузки.

3.2. Часть 2: описание схемы в Vivado

В соответствии с заданием был разработан программный модуль описывающий работу БОЭ (декодера 3 в 8), см. листинг 1.1. При его реализации

в качестве базисных логических элементов так же использовались NAND компонента, но на в этом случае программная среда предоставляет его без необходимости реализовывать самому. В добавок к этому, можно наблюдать значительное отличие метода описания функциональной схемы кодом от составления графической схемы — программная среда сама заботится о передаче в компоненты низкого и высокого логического уровня сигнала, поэтому элементы `nand` имеют всего 3 аргумента: выходной порт и два ВХОДНЫХ.

```

1  'timescale 1ns / 1ps
2
3  module decoder(
4      input [2:0] s,
5      input en,
6      output [7:0] d
7  );
8      wire [2:0] not_s;
9      wire [7:0] not_s_2_1;
10     wire [7:0] not_s_0_en;
11     wire [7:0] s_2_1;
12     wire [7:0] s_0_en;
13     wire [7:0] not_d;
14
15     nand(not_s[2], s[2], s[2]);
16     nand(not_s[1], s[1], s[1]);
17     nand(not_s[0], s[0], s[0]);
18
19     nand(not_s_2_1[0], not_s[2], not_s[1]);
20     nand(not_s_0_en[0], not_s[0], en);
21     nand(s_2_1[0], not_s_2_1[0], not_s_2_1[0]);
22     nand(s_0_en[0], not_s_0_en[0], not_s_0_en[0]);
23     nand(not_d[0], s_2_1[0], s_0_en[0]);
24
25     nand(not_s_2_1[1], not_s[2], not_s[1]);
26     nand(not_s_0_en[1], s[0], en);
27     nand(s_2_1[1], not_s_2_1[1], not_s_2_1[1]);
28     nand(s_0_en[1], not_s_0_en[1], not_s_0_en[1]);
29     nand(not_d[1], s_2_1[1], s_0_en[1]);
30
31     nand(not_s_2_1[2], not_s[2], s[1]);
32     nand(not_s_0_en[2], not_s[0], en);
33     nand(s_2_1[2], not_s_2_1[2], not_s_2_1[2]);
34     nand(s_0_en[2], not_s_0_en[2], not_s_0_en[2]);
35     nand(not_d[2], s_2_1[2], s_0_en[2]);
36
37     nand(not_s_2_1[3], not_s[2], s[1]);
38     nand(not_s_0_en[3], s[0], en);
39     nand(s_2_1[3], not_s_2_1[3], not_s_2_1[3]);
40     nand(s_0_en[3], not_s_0_en[3], not_s_0_en[3]);
41     nand(not_d[3], s_2_1[3], s_0_en[3]);
42
43     nand(not_s_2_1[4], s[2], not_s[1]);
44     nand(not_s_0_en[4], not_s[0], en);
45     nand(s_2_1[4], not_s_2_1[4], not_s_2_1[4]);
46     nand(s_0_en[4], not_s_0_en[4], not_s_0_en[4]);
47     nand(not_d[4], s_2_1[4], s_0_en[4]);
48

```

```

49     nand(not_s_2_1[5], s[2], not_s[1]);
50     nand(not_s_0_en[5], s[0], en);
51     nand(s_2_1[5], not_s_2_1[5], not_s_2_1[5]);
52     nand(s_0_en[5], not_s_0_en[5], not_s_0_en[5]);
53     nand(not_d[5], s_2_1[5], s_0_en[5]);
54
55     nand(not_s_2_1[6], s[2], s[1]);
56     nand(not_s_0_en[6], not_s[0], en);
57     nand(s_2_1[6], not_s_2_1[6], not_s_2_1[6]);
58     nand(s_0_en[6], not_s_0_en[6], not_s_0_en[6]);
59     nand(not_d[6], s_2_1[6], s_0_en[6]);
60
61     nand(not_s_2_1[7], s[2], s[1]);
62     nand(not_s_0_en[7], s[0], en);
63     nand(s_2_1[7], not_s_2_1[7], not_s_2_1[7]);
64     nand(s_0_en[7], not_s_0_en[7], not_s_0_en[7]);
65     nand(not_d[7], s_2_1[7], s_0_en[7]);
66
67     nand(d[0], not_d[0], not_d[0]);
68     nand(d[1], not_d[1], not_d[1]);
69     nand(d[2], not_d[2], not_d[2]);
70     nand(d[3], not_d[3], not_d[3]);
71     nand(d[4], not_d[4], not_d[4]);
72     nand(d[5], not_d[5], not_d[5]);
73     nand(d[6], not_d[6], not_d[6]);
74     nand(d[7], not_d[7], not_d[7]);
75
76 endmodule

```

Листинг 1.1: Код программного модуля реализующего логику требуемого БОЭ, декодера 3 в 8

Также в соответствии с заданием был разработан программный модуль, тестирующий логику ранее разработанного БОЭ, см. листинг 1.2. В нем в цикле для всех возможных входных значений сигналов, соответствующих двоичному представлению чисел от 0 до 7, проверяется, что на выход подаются сигналы, соответствующие двоичному представлению числа 2 в степени входного двоичного числа. Иными словами, логическая 1 должна быть выставлена на порт с номером, равным 2 в степени входного двоичного числа. В случае соответствия работы компонента поставленным требованиям, на экран выводится сообщение о корректности работы для конкретной комбинации сигналов; в противном случае — сообщение о некорректности работы.

```

1  'timescale 1ns / 1ps
2
3  module decoder_tb;
4
5      reg [2:0] s;
6      wire [7:0] d;
7      reg en;
8
9      integer i;
10
11      decoder decoder_1(

```


```

12     .s(s),
13     .d(d),
14     .en(en)
15 );
16
17 initial begin
18     for(i = 0; i < 8; i = i+1) begin
19         s = i;
20         en = 1;
21
22         #10
23
24         if (d == 2**i) begin
25             $display("Correct! s=%b, d=%b, en=%b, i=%0d", s, d, en, i);
26         end else begin
27             $display("Incorrect! s=%b, d=%b, en=%b, i=%0d", s, d, en, i);
28         end
29
30         en = 0;
31         #10
32
33         if (d == 0) begin
34             $display("Correct! s=%b, d=%b, en=%b, i=%0d", s, d, en, i);
35         end else begin
36             $display("Incorrect! s=%b, d=%b, en=%b, i=%0d", s, d, en, i);
37         end
38
39     end
40     #10 $stop;
41
42 end
43 endmodule

```

Листинг 1.2: Код программного модуля тестирующего логику позиционного декодера 3 в 8

Еще в среде разработки Vivado была получена временная диаграмма симуляции работы разработанного компонента, её можно видеть на изображении



res/3-to-8-decoder_vivado-simulation.png

Рис. 1.12: Временная диаграмма симуляции работы БОЭ в среде Vivado

4. Вывод

В рамках выполнения работы были выполнены все поставленные задачи. Были приобретены навыки разработки цифровых схем в приложении LTSpice с последующей симуляцией их работы. Главной сложностью при этом стало понять логику комбинирования функциональных компонентов: для меня было не очевидно, что сигнал «питания» и сигнал высокого логического уровня не совпадают. Оказалось, что для более точной физической реализации, каждый компонент должен быть подключен независимо к линии с высоким логическим уровнем напряжения. Также было видно, насколько выбор логического базиса влияет на сложность составления БОЭ, мне удалось наблюдать, как другим учащимся было тяжело составить свою схему с помощью других базисных элементов.

В то же время, среда разработки Vivado и в частности язык Verilog позволяют избавиться от муторного и сложного управления подключением линий сигнала логических 1 и 0. Вместо этого Verilog позволяет описывать логику взаимодействия компонентов без погружения в детали физического

устройства компонентов.

Отдельное впечатление оставляет способ дистрибуции среды Vivado: 20 (ГБ) образ для версии 2019 года — это что-то с чем-то. Более того на сайте производителя эта система поставляется образами по 100 с лишним гигабайт, что заставляет задуматься о внутреннем устройстве этой системы, ведь внешний вид у нее не блещет изяществом, и требованиям к средствам разработки реальных вычислительных систем. Вероятно это приложение содержит в себе огромное количество дополнительных драйверов для работы с разными типами устройств.