ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

РАСЧЁТНО-ГРАФИЧЕСКАЯ РАБОТА

по дисциплине “Схемотехника”

Выполнил студент Ермаков Арсений Владимирович
Ф.И.О.

Группы ИВ-121

Работу принял _____ Максимов И. Н.
подпись

Защищена _____ Оценка _____

Новосибирск – 2023

Содержание

Задание

Необходимо создать проект, в котором нужно реализовать схему возведения целого числа в пятую степень (рис. 1.1).

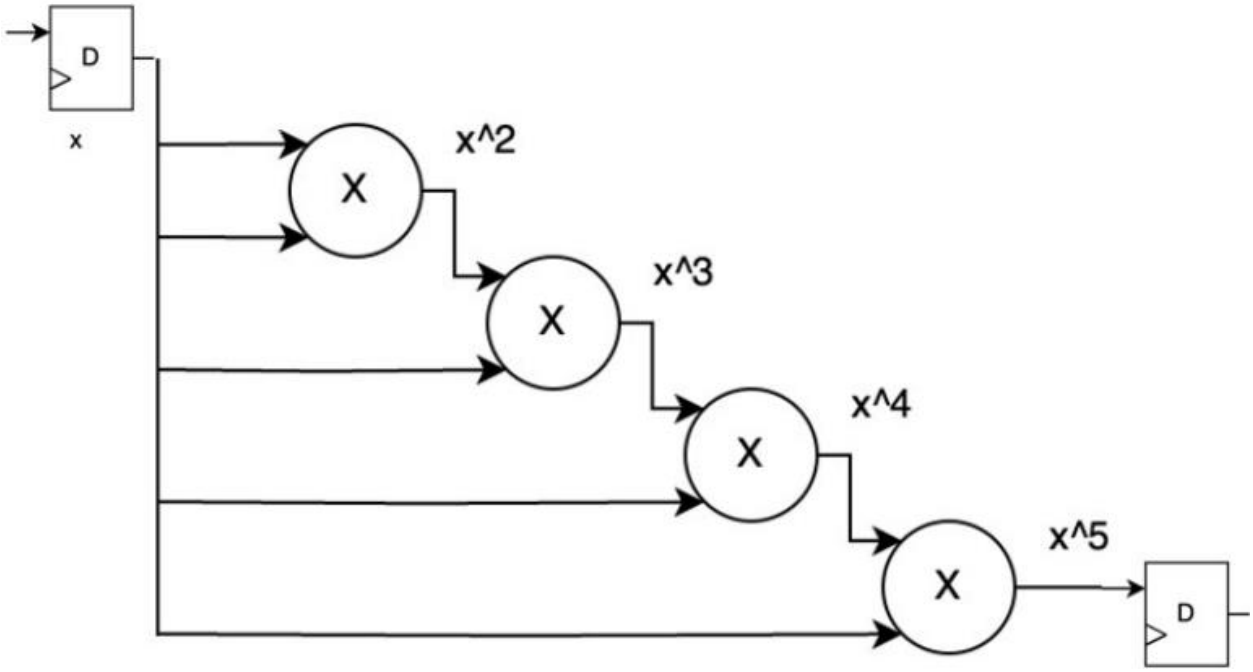

Рис. 1.1 – схема возведения числа в пятую степень

При этом разрядность чисел нужно выбрать произвольно, но не меньше шестнадцати.

По результатам работы схемы определить максимальную тактовую частоту, которая представлена в программе Quartus по следующему пути:

Compilation Report -> Timing Analyzer -> Slow 1200mV 85c Model -> Fmax Summary

Рис. 1.2 Путь до файла с информацией о частоте

Вторым этапом работы будет добавление в схему конвейеризации (рис. 1.3).

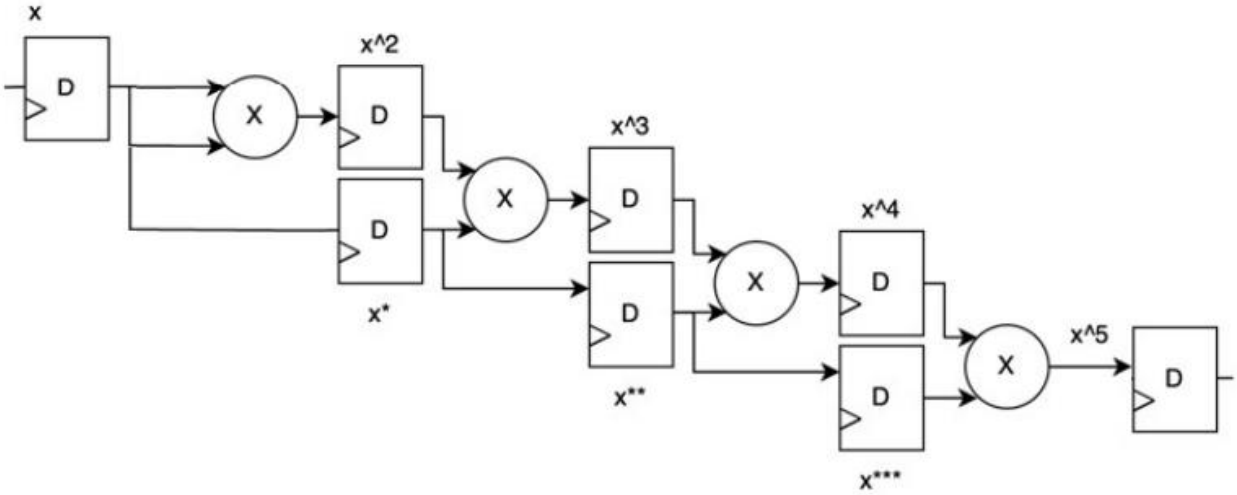

Рис. 1.3 Схема возведения числа в пятую степень с конвейеризацией

На данном этапе необходимо определить максимальную тактовую частоту и ответить на вопрос: «Стала ли максимальная частота больше?».

В заключение нужно ответить на вопрос: «Зависят ли полученные частоты от разрядности чисел, с которыми оперирует схема?» и подтвердить свой ответ практическим результатом.

1. Схема без конвейеризации

1.1 Реализация кода

Для начала реализуем схему возведения числа в пятую степень без конвейеризации (рис. 1).
Для этого напишем модуль **degree**, в котором зададим параметр **DIGIT** (разряд числа) равный 32 (по заданию можно брать любое количество, но больше 16), а также тактовый сигнал **clk**, входной сигнал **number** и выходной – **out_number** (рис. 2.1).

```
1  module degree #(parameter DIGIT = 32)(
2   input logic clk,
3   input logic [DIGIT-1:0] number,
4   output logic [?] out_number
5  );
```

Рис. 2.1 Интерфейс модуля degree

Однако сразу возникает вопрос: «Какую разрядность должен иметь выходной сигнал?». Для того чтобы ответить на этот вопрос, мы посчитаем максимальную разрядность числа после возведения его во 2, 3, 4 и 5 степени. Представим максимальное 32-х разрядное число в десятичной системе счисления как $2^{32} – 1$ и будем последовательно домножать его на него же:

$$(2^{32} – 1) \times (2^{32} – 1) = 2^{64} – 2^{33} + 1$$

Формула. 2.1 Разложение возведенного во 2-ю степень 32-х разрядного числа

Из формулы 2.1 видно, что максимальная степень числа 2 в разложении будет 64, следовательно максимальная разрядность полученного числа будет 64 (с учетом вычитания 2^{33}), а это в свою очередь значит, что в последующих домножениях на 32-х разрядное число степень полученного числа будет увеличиваться на 32 разряда после каждой операции (табл. 2.1).

Число на этапе домножения	Результат умножения на $2^{32} – 1$	Степень возведения исходного числа	Разрядность
$(2^{32} - 1)^1$	$2^{64} – 2^{33} + 1$	2	64
$(2^{32} - 1)^2$	$2^{96} - 3 \times 2^{64} + 3 \times 2^{32} - 1$	3	96
$(2^{32} - 1)^3$	$2^{128} - 2^{98} + 3 \times 2^{65} - 2^{34} + 1$	4	128
$(2^{32} - 1)^4$	$2^{160} + 5(-2^{128} + 2^{97} - 2^{65} + 2^{32}) - 1$	5	160

Таблица. 2.1 Зависимость разрядности числа от степени возведения

Таким образом, мы получили необходимые разрядности чисел для каждой

операции умножения на $2^{32} - 1$, и теперь для выходного сигнала разрядность будет 160 или 32×5:

```
1  output [DIGIT*5-1:0] out_num
```

Рис. 2.2 Выходной сигнал

Далее, пользуясь полученной таблицей (табл. 2.1), определим три переменные: **num**, которая инициализируется числом **number** при положительном фронте тактового сигнала **clk**, содержит исходное число и используется как операнд для последующих операций, **out_num**, которая также инициализируется, как и **num**, содержит результат возведения числа в пятую степень, и используется для вывода через **out_number** и, наконец, **cur**, которая накапливает в себе результаты произведений:

```
 1  logic [DIGIT-1:0] num;
 2  logic [DIGIT*5-1:0] out_num;
 3  logic [DIGIT*5-1:0] cur;
 4  always @(posedge clk)
 5   num <= number;
 6  //
 7  // блок умножений
 8  //
 9  always @(posedge clk)
10   out_num <= cur;
11  assign out_number = out_num;
12  endmodule
```

Рис. 2.3 Использование необходимых переменных

Последнее, что необходимо выполнить – это реализовать комбинаторный логический блок кода с произведениями, который будет возводить число в пятую степень. Для этого будем использовать **always_comb**:

```
1  always_comb
2   begin
3    cur = num * num;
4    cur *= num;
5    cur *= num;
6    cur *= num;
7   end
```

Рис. 2.4 Логический блок с произведениями

1.2 Результаты

После успешной компиляции кода в программе Quartus через RTL viewer построим схему возведения числа в пятую степень без конвейеризации:

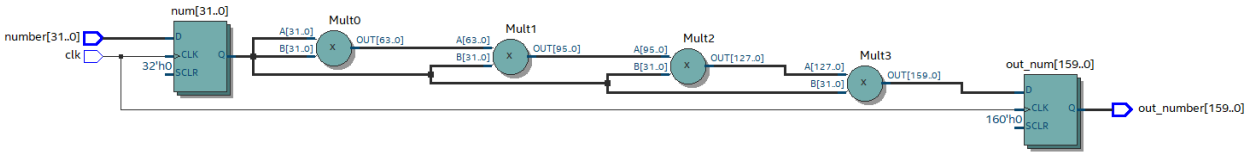

Рис. 3.1 Построенная схема без конвейеризации

Можно заметить, что полученная схема полностью совпадает со схемой, которая была приведена в задании (рис. 1.1).

Перейдем в нужный нам файл по пути, представленному в задании (рис. 1.2), и посмотрим максимальную тактовую частоту:

	Fmax	Restricted Fmax	Clock Name	Note
1	23.29 MHz	23.29 MHz	clk	

Рис. 3.2 Максимальная частота без конвейеризации

Получившаяся частота достаточно низкая, потому необходимо ее повысить с помощью добавления конвейеризации.

2. Схема с конвейеризацией

2.1 Реализация кода

Будем использовать такой же интерфейс, как у модуля без конвейеризации, оставив его без изменений (рис. 2.1). Определим необходимые переменные (регистры) для хранения промежуточных результатов на различных стадиях конвейера, а также для последовательных произведений для возведения числа в пятую степень, при это будем использовать таблицу 2.1:

```
1   logic [DIGIT-1:0] R1;
2   logic [DIGIT*2-1:0] R2;
3   logic [DIGIT*3-1:0] R3;
4   logic [DIGIT*4-1:0] R4;
5   logic [DIGIT*5-1:0] R5;
6   logic [DIGIT-1:0] res1, res2, res3;
```

Рис. 4.1 Переменные кода с конвейеризацией

В этом коде мы будем использовать **always_ff**, чтобы обеспечить гарантии по времени и стабильности, что в свою очередь является более надежным подходом, чем использование обычного **always**.

Далее реализуем блок кода с конвейеризацией, схема которого была представлена в задании на соответствующем рисунке (рис. 1.3):

```
 1      R1 <= number;
 2      R2 <= R1 * R1;
 3      res1 <= R1;
 4
 5      R3 <= R2 * res1;
 6      res2 <= res1;
 7
 8      R4 <= R3 * res2;
 9      res3 <= res2;
10
11      R5 <= R4 * res3;
```

Рис. 4.2 Блок с конвейеризацией

В данном коде конвейеризация реализована за счёт последовательного использования регистров для хранения промежуточных результатов вычислений на каждом такте. Рассмотрим ее на примере написанного кода для каждого этапа:

1) **R1** сохраняет значение **number** на восходящем фронте тактового сигнала.
2) **R2** сохраняет результат умножения **R1** * **R1**, который представляет собой изначальное **number**, возведенное во вторую степень.
3) R3 сохраняет результат умножения **R2** * **res1**, который представляет **number** в третьей степени. **res1** хранит значение **R1**, зафиксированное на предыдущем такте, что позволяет **R3** вычислять $\mathbf{R1^3}$ независимо от

текущего значения **R1**.

4) Аналогично, **R4** сохраняет результат умножения **R3** * **res2**, представляющий number в четвертой степени, и **R5** — number в пятой степени, используя **R4** * **res3**.

Регистры **res1**, **res2**, **res3** действуют как буферы, позволяя каждому этапу умножения работать с данными, которые были захвачены на предыдущем такте. Это позволяет умножениям выполняться параллельно в разных частях схемы, что увеличивает пропускную способность и позволяет схеме работать на более высокой тактовой частоте.

2.1 Результаты

После успешной компиляции кода с конвейеризацией, как и в предыдущем случае построим схему:

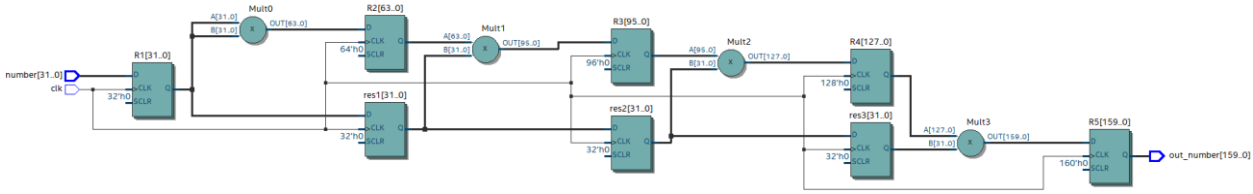

Рис. 5.1 Построенная схема с конвейеризацией

Нетрудно заметить, что построенная схема совпадает полностью со схемой, приведенной в задании (рис. 1.3).

Теперь, как и в случае без конвейеризации, перейдем в нужный файл по пути, представленному в задании (рис. 1.2) и посмотрим максимальную тактовую частоту:

	Fmax	Restricted Fmax	Clock Name	Note
1	56.29 MHz	56.29 MHz	clk	

Рис. 5.2 Максимальная частота с конвейеризацией

В итоге мы получили частоту **Fmax** = 56.29 MHz, которая значительно выше частоты без конвейеризации:

$$\frac{56.29}{23.29} \approx 2.4$$

Рис. 5.3 коэффициент увеличения частоты

Таким образом при использовании конвейеризации для возведения 32-х разрядного числа в пятую степень (даже при использовании огромных 64, 96, 128 и 160-ти разрядных регистров) получился прирост более чем в 2.4 раза, что является неплохим результатом.

Тестирование кода

Мы написали необходимые модули, построили схемы (рис. 3.1 и рис. 5.1), получили максимальные тактовые частоты. Однако любой написанный код требует тестирования, поэтому напишем общий модуль тестирования для кода без конвейеризации и кода с конвейеризацией:

```
1  `timescale 1ns/1ns
2
3  module testbench();
4    parameter DIGIT = 32;
5
6    logic clk;
7    logic [DIGIT-1:0] number;
8    logic [DIGIT*5-1:0] out_number;
9
10   degree #(DIGIT) dut (
11     .clk(clk),
12     .number(number),
13     .out_number(out_number)
14   );
15
16   initial begin
17     clk = 0;
18     forever #5 clk = ~clk;
19   end
20
21   initial begin
22     number = 32'b11111111111111111111111111111111;
23     repeat (5) begin
24       #50;
25     end
26
27     $display("Result = %h", out_number);
28   end
29
30 endmodule
```

Листинг 6.1 Тестирующий модуль (без конвейеризации)

В данном модуле, как и в самом коде, мы будем использовать параметр **DIGIT** равным 32. Именно поэтому входное число имеет максимальное значение 32 единицы:

Рис. 6.1 Число $FFFFFFFF_{16}$ в двоичной системе счисления

Теперь посчитаем, каким должен быть результат возведения $FFFFFFFF_{16}$ в пятую степень, для этого переведем исходное число в десятичную систему счисления, получим 4294967295_{10} и выполним на калькуляторе операцию возведения числа в пятую степень:

$$4\,294\,967\,295^5$$

Result

1 461 501 635 629 491 084 391 274 140 357 585 917 716 910 309 375

Рис. 6.2 Число 4294967295_{10} в пятой степени

Далее результат переведем в 16-ричную систему счисления и получим FFFFFFFB00000009FFFFFFF600000004FFFFFFFF$_{16}$.
Теперь компилируем, запускаем тестирующий модуль и смотрим вывод:

```
VSIM 49> run -all
# Result = fffffffb00000009fffffff600000004ffffffff
```

Рис. 6.3 Результат возведения FFFFFFFF$_{16}$ в пятую степень

Нетрудно заметить, что результаты совпадают, а значит программа корректно работает.

В случае тестирования кода с конвейеризацией были использованы все тоже 32-х разрядное число FFFFFFFF$_{16}$ и тот же тестирующий модуль (лист. 6.1). Результаты оказались аналогичными предыдущим (рис. 6.3).

Изменение числа разрядов

Теперь нам нужно ответить на вопрос, представленный в задании: «Как вы думаете, зависят ли полученные частоты от разрядности чисел, с которыми оперирует схема?», а точнее дать теоретический ответ и подкрепить его практическим результатом.

Достаточно очевидно, что количество бит числа, которые при умножении обрабатываются, влияет на производительность и тактовую частоту, а это в свою очередь значит, что чем больше разрядов у числа, тем большим количеством бит придется оперировать, а это лишь означает, что чем меньше разрядов у числа, тем выше тактовая частота, а чем больше разрядность числа, тем частота ниже.

В коде мы использовали 32-х разрядное число, из-за которого количество разрядов результирующего регистра составляло 160 (табл. 2.1), поэтому для примера возьмем следующие исходные числа: 8-ми, 16-ти, 24-х, 40-ка, 48-ми, 56-ти и 64-х разрядные, изменим семь раз значение параметра DIGIT в каждом коде и посмотрим, как будет меняться максимальная частота.

Для начала, используя полученный анализ разрядности всех регистров (табл. 2.1), вычислим разрядности регистров каждого входного числа для каждой степени его возведения:

Степень возведения числа	Разрядность регистров							
1 (исходное число)	8	16	24	32	40	48	56	64
2	16	32	48	64	80	96	112	128
3	24	48	72	96	120	144	168	192
4	32	64	96	128	160	192	224	256
5	40	80	120	160	200	240	280	320

Таблица 7.1 Зависимость разрядностей от степени возведения числа

Из таблицы видно, что начальные регистры различаются между собой на 8

разрядов, что не много по сравнению с результирующими регистрами, которые различаются уже на 40 (!) разрядов, а это лишь означает, что частоты будут существенно отличаться.

На базе экспериментов построим еще одну таблицу, которая показывает зависимость разрядности исходного числа от максимальной тактовой частоты:

Использование конвейеризации	Разрядность исходного числа							
	8	16	24	32	40	48	56	64
	Соответствующая максимальная тактовая частота							
Нет	53.06	39.63	27.02	23.29	20.8	18.9	18.23	17.29
Да	161.21	123.96	74.96	56.29	55.14	42.17	45.99	37.85

Таблица 7.2 Полученные максимальные тактовые частоты

При просмотре в таблице 7.2 строк, содержащих максимальные частоты без использования конвейеризации, можно заметить, что тактовая частота падает с увеличением разрядности исходного числа.

Если же глянуть в строку, содержащую максимальные частоты при условии использования конвейеризации, то картина в целом та же, только разница между максимальными тактовыми частотами при использовании 32-х и 40-ка разрядных исходных чисел невелика, а вот тактовые частоты при использовании 48-ми и 56-ти разрядных исходных чисел странно отличаются: частота при использовании 56 разрядного числа выше (!), чем частота при использовании 48-ми разрядного числа. Скорее всего на это влияет как сама конвейеризация, так и работа конкретного анализатора частоты (рис. 1.2).

Для больше наглядности по полученным данным (табл. 7.2) с помощью языка программирования Python построим столбчатую диаграмму, показывающую зависимость максимальной тактовой частоты от разрядности числа:

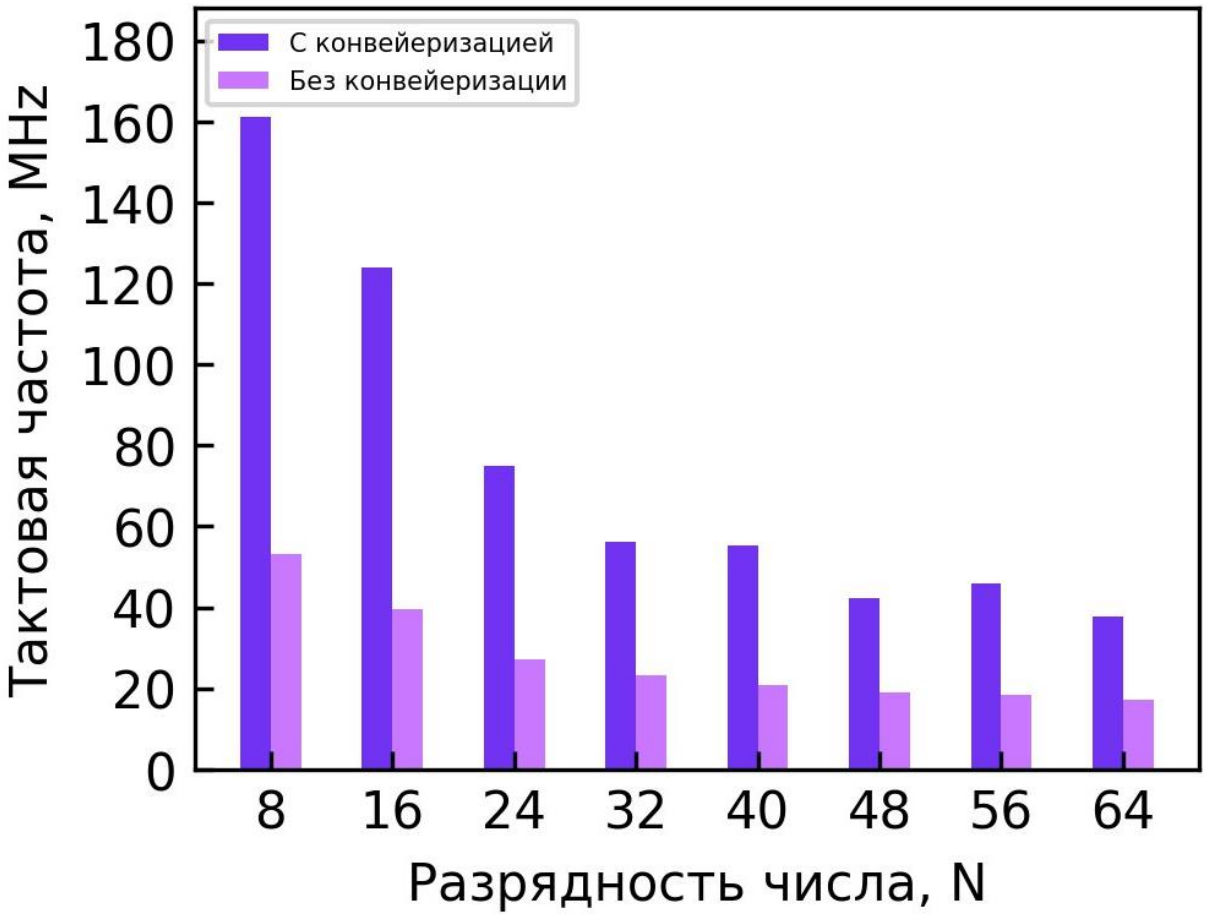

Рис. 7.1 Столбчатая диаграмма масштабируемости

Для 8-ми и 16-ти разрядных чисел тактовые частоты гораздо выше, чем для чисел с большими разрядностями, поэтому построим еще одну аналогичную столбчатую диаграмму для чисел, имеющих от 24 до 64 разрядов:

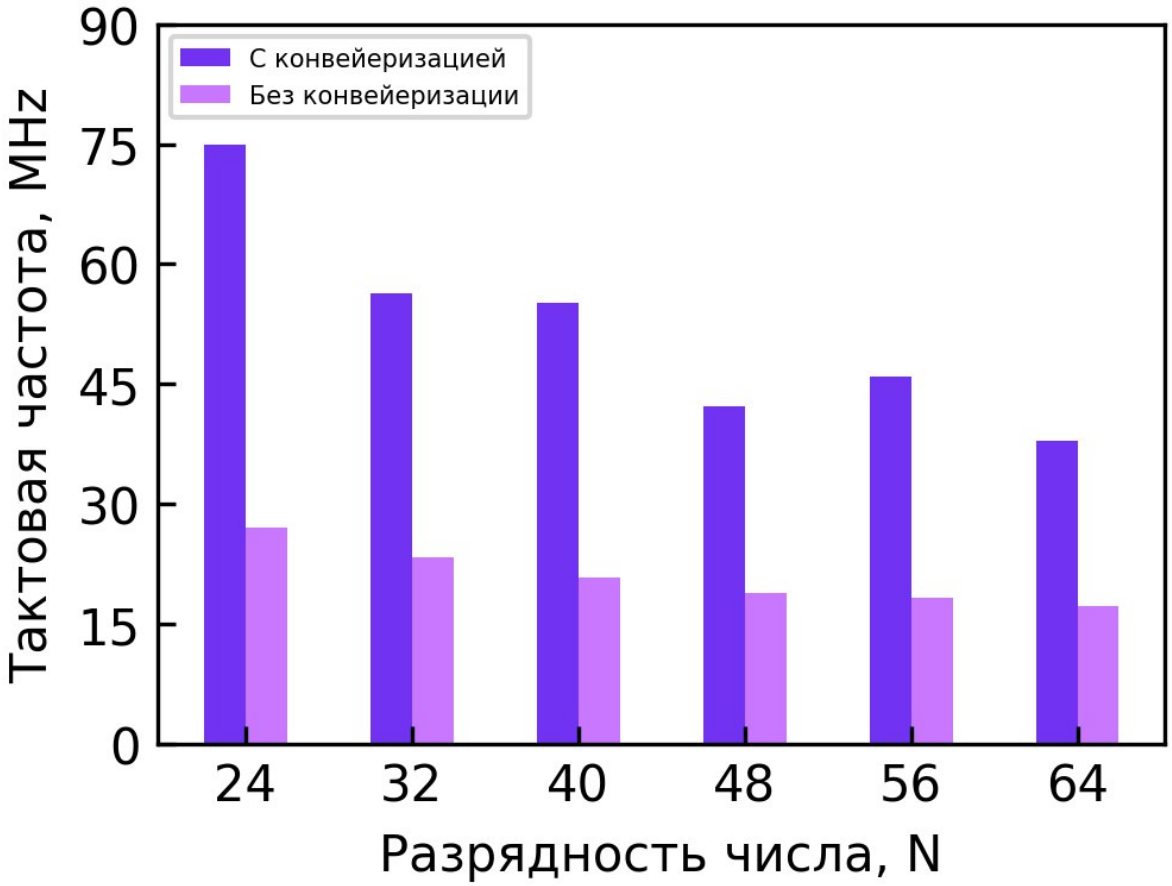

Рис. 7.2 Столбчатая диаграмма масштабируемости (разрядности от 24 до 64)

По полученным результатам (табл. 7.2) и построенным столбчатым диаграммам (рис. 7.1 и рис. 7.2) можно сделать следующие выводы:

1. Теоретическая оценка оказалась верной.
2. Практический результат подтвердил теоретическую оценку.
3. Полученные частоты зависят от разрядности чисел, с которыми оперирует схема.
4. Чем больше разрядность числа, тем меньше максимальная тактовая частота.

Заключение

В ходе выполнения работы мы написали модули **degree** без конвейеризации и с конвейеризацией, получили максимальные тактовые частоты для каждого из модулей, написали тестирующий модуль и проверили корректность работы написанного кода.

Далее мы несколько раз изменяли разрядность входного числа, получая соответствующие ему частоты для обоих модулей **degree**, построили по полученным данным столбчатые диаграммы и сделали вывод, что уменьшение разрядности входного числа повышает максимальную тактовую частоту, как и использование конвейеризации в коде.

Список использованных источников

1. Дэвид М. Хэррис и Сара Л. Хэррис Цифровая схемотехника и архитектура компьютера. - 2 изд. - New York: Morgan Kaufman, 2013. - 1684 с.
2. wikipedia : сайт. – URL: https://ru.wikipedia.org/wiki/Вычислительный_конвейер (дата обращения: 26.12.2023)
3. habr : сайт. – URL: https://habr.com/ru/articles/182002/ (дата обращения: 26.12.2023)
4. studfile : сайт. – URL: https://studfile.net/preview/4420936/page:19/ (дата обращения: 26.12.2023)
5. wolframalpha : сайт. – URL: https://www.wolframalpha.com/input?i=4294967295^5 (дата обращения: 26.12.2023)
6. leonid-korobkov : сайт. – URL: https://leonid-korobkov.github.io/math-project/ (дата обращения: 26.12.2023)

Листинг кода

Модуль **degree** без конвейеризации:

```
module degree #(parameter DIGIT = 32)(
 input logic clk,
 input logic [DIGIT-1:0] number,
 output logic [DIGIT*5-1:0] out_number
);
logic [DIGIT-1:0] num;
logic [DIGIT*5-1:0] out_num;
logic [DIGIT*5-1:0] cur;

always @(posedge clk)
 num <= number;

always_comb
 begin
  cur = num * num;
  cur *= num;
  cur *= num;
  cur *= num;
 end

always @(posedge clk)
 out_num <= cur;

assign out_number = out_num;
endmodule
```

Модуль **degree** с конвейеризацией:

```
module degree #(parameter DIGIT = 32)(
  input logic clk,
  input logic [DIGIT-1:0] number,
  output logic [DIGIT*5-1:0] out_number
);

  logic [DIGIT-1:0] R1;
  logic [DIGIT*2-1:0] R2;
  logic [DIGIT*3-1:0] R3;
  logic [DIGIT*4-1:0] R4;
  logic [DIGIT*5-1:0] R5;
  logic [DIGIT-1:0] res1, res2, res3;

  always_ff @(posedge clk)
  begin
    R1 <= number;
    R2 <= R1 * R1;
    res1 <= R1;

    R3 <= R2 * res1;
    res2 <= res1;

    R4 <= R3 * res2;
    res3 <= res2;
```

```
25
26      R5 <= R4 * res3;
27    end
28
29    assign out_number = R5;
30  endmodule
```