Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчёт по лабораторной работе 5s**

Дисциплина: Языки описания аппаратных средств вычислительных систем.

Выполнил студент гр. 5130901/10101 \_\_\_\_\_\_\_\_\_\_\_\_\_ М. Т. Непомнящий

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Федотов

(подпись)

Санкт-Петербург

2023

Оглавление

[1. Задача: 3](file:///D:\Users\Legion\Downloads\Отчет%20по%20Verilog.%20Симоновский%20Д.Л.%20гр.%205130901_10101.docx#_Toc151760250)

[2. Решение: 3](file:///D:\Users\Legion\Downloads\Отчет%20по%20Verilog.%20Симоновский%20Д.Л.%20гр.%205130901_10101.docx#_Toc151760251)

[3. Вывод: 13](file:///D:\Users\Legion\Downloads\Отчет%20по%20Verilog.%20Симоновский%20Д.Л.%20гр.%205130901_10101.docx#_Toc151760252)

# Задача:

На языке Verilog разработать:

* Параметризированный умножитель, реализующий умножение [W-1:0] dA на [W-1:0] dB по алгоритму «умножение старшими разрядами вперед со сдвигом суммы».
* Реализовать потоковой умножитель т.е. вычисления выполняются на комбинационной схеме, реализованной как сеть связанных друг с другом сумматоров и мультиплексоров, имеющей на входе и выходе регистры.
* Разработать тест 2 класса для всех возможных вариантов входных данных.

***Модуль mult\_L\_V1:***

**Входы:**

* сlk – тактовый сигнал.
* [W-1:0] dA – вход данных A.
* [W-1:0] dB – вход данных B.

**Выходы:**

* [2\*W-1:0] a\_mult\_b – результат умножения.

На языке Verilog разработать:

* Конвейеризированный вариант модуля mult\_L\_V1.
* Разработать тест

***Модуль mult\_LP\_V1:***

**Входы:**

* сlk – тактовый сигнал.
* [W-1:0] dA – вход данных A.
* [W-1:0] dB – вход данных B.

**Выходы:**

* [2\*W-1:0] a\_mult\_b – результат умножения.

# Решение:

Алгоритм умножителя старшими разрядами вперед выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рис. 2.1 – Алгоритм умножения старшими разрядами вперед

Рассмотрим пример работы умножителя, представленный в методическом пособии:

Изображение выглядит как текст, число, Шрифт, Параллельный

Автоматически созданное описание

Рис. 2.2 – Алгоритм умножения старшими разрядами вперед

Данный алгоритм имеет два этапа вычислений: первый – сдвиг, второй – суммирование.

Для реализации комбинационной схемы необходимо использовать блок generate, на каждой итерации будем выполнять сдвиг результата и добавлять A, если разряд B равен 1. Будем перебирать B с конца, как показано в алгоритме. Итоговый код представлен на рисунке ниже:

**module** mult\_L\_V1 **#(**

**parameter** W **=** 8

**)** **(**

**input** clk**,**

**input** **[** W**-**1**:**0**]** dA**,**

**input** **[** W**-**1**:**0**]** dB**,**

**output** **reg** **[**2**\***W**-**1**:**0**]** a\_mult\_b

**);**

**reg** **[**2**\***W**-**1**:**0**]** dSUM**[**0**:**W**];**

**reg** **[**W**-**1**:**0**]** A**,** B**;**

**initial** **begin**

**for** **(int** i **=** 0**;** i **<=** W**;** i**++)** dSUM**[**i**]** **<=** 1'b0**;**

A **<=** 1'b0**;**

B **<=** 1'b0**;**

a\_mult\_b **<=** 1'b0**;**

**end**

**always** **@(posedge** clk**)** **begin**

A **<=** dA**;**

B **<=** dB**;**

**end**

**genvar** i**;**

**generate**

**for** **(**i **=** 0**;** i **<** W**;** i**++)** **begin** **:** generator

**always** **@\*** dSUM**[**i**+**1**]** **=** **(**dSUM**[**i**]** **<<** 1'b1**)** **+** **(**B**[**W**-**i**-**1**]** **?** A **:** 1'b0**);**

**end**

**endgenerate**

**always** **@(posedge** clk**)** a\_mult\_b **<=** dSUM**[**W**];**

**endmodule**

Выполним моделирование разработанного модуля. Для этого разработаем следующий тестовый модуль, который перебирает все варианты входных параметров:

`timescale 1ns **/** 1ns

**module** mult\_L\_V1\_tb**;**

**parameter** PERIOD **=** 10**;**

**parameter** W **=** 8**;**

**reg** clk **=** 0**;**

**reg** **[** W**-**1**:**0**]** dA **=** 0**;**

**reg** **[** W**-**1**:**0**]** dB **=** 0**;**

**wire** **[**2**\***W**-**1**:**0**]** a\_mult\_b**;**

**initial** **begin**

**forever** **#(**PERIOD **/** 2**)** clk **=** **~**clk**;**

**end**

mult\_L\_V1 **#(**

**.**W**(**W**)**

**)** u\_mult\_L\_V1 **(**

**.**clk**(**clk**),**

**.**dA **(**dA**[**W**-**1**:**0**]),**

**.**dB **(**dB**[**W**-**1**:**0**]),**

**.**a\_mult\_b**(**a\_mult\_b**[**2**\***W**-**1**:**0**])**

**);**

**initial** **begin**

**for** **(int** i **=** 0**;** i **<** $pow**(**2**,** W**);** i**++)**

**for** **(int** j **=** 0**;** j **<** $pow**(**2**,** W**);** j**++)** **begin**

dA **<=** i**;**

dB **<=** j**;**

**#(**PERIOD **\*** 2**);**

**if** **(**i **\*** j **!=** a\_mult\_b**)** **begin**

$display**(**"Incorrect result: %0d \* %0d = %0d\nExpected: %0d"**,** i**,** j**,** a\_mult\_b**,** i **\*** j**);**

$stop**;**

**end**

**end**

$display**(**"All tests passed!"**);**

$stop**;**

**end**

**endmodule**

Запустим моделирование в ModelSim (при этом зададим параметр W = 8). \*При открытии Wave визуально может быть не понятно, что и где находится (рис. 2.4): это связано с большим количеством значении, при приближении всё становится понятно.

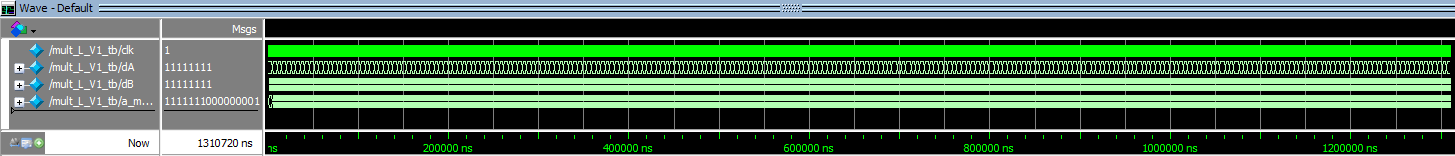


Рис. 2.3 – Wave для mult\_L\_V1\_tb

Благодаря сообщения, выведенному на консоль по окончании компиляции можем удостовериться в правильности работы программы:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

Рис. 2.4 – Transcipt для mult\_L\_V1\_tb

Выполним компиляцию модуля mult\_L\_V1 в среде Quartus, с параметром W = 4:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рис. 2.5 – Результат компиляции mult\_L\_V1 с W = 4

Также посмотрим на максимальную частоту устройства:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рис. 2.6 – Максимальная частота устройства mult\_L\_V1 с W = 4

Повторим скриншоты с W = 8 и W = 16:

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рис. 2.7 – Результат компиляции mult\_L\_V1 с W = 8

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рис. 2.8 – Максимальная частота устройства mult\_L\_V1 с W = 8

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рис. 2.9 – Результат компиляции mult\_L\_V1 с W = 16

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рис. 2.10 – Максимальная частота устройства mult\_L\_V1 с W = 16

Из полученных данных можно сделать следующий вывод: чем больше разрядность, чем больше затрачиваемые ресурсы и меньше максимальная частота устройства.

Рассмотрим RTL Viewer разработанного устройства. Для удобства анализа поставим W = 4 (удобное число элементов цепи, позволяющее не слишком сильно «забить» изображение, но, при этом, достаточное, чтобы показать принцип работы блоков):

Изображение выглядит как диаграмма, линия, План

Автоматически созданное описание

Рис. 2.11 – RTL Viwer для mult\_L\_V1

Как видно, каждая итерация из двух блоков MUX и 1 блока ADD. Оба MUX используются для того, чтоб определить, делать ли сумму или нет. Один из MUX отвечает за младший разряд т. к. после сдвига итоговой суммы его можно записать сразу же, остальные же разряды идут во второй MUX, который формирует число для суммы, уже использованный младший разряд замещается нулем при суммировании.

Дабы повысить максимальную частоту устройства, выполним конвейеризацию:

**module** mult\_LP\_V1 **#(**

**parameter** W **=** 4

**)** **(**

**input** clk**,** // Входной тактовый сигнал

**input** **[** W**-**1**:**0**]** dA**,** // Входной операнд A

**input** **[** W**-**1**:**0**]** dB**,** // Входной операнд B

**output** **reg** **[**2**\***W**-**1**:**0**]** a\_mult\_b // Выход, хранящий результат умножения

**);**

**reg** **[**2**\***W**-**1**:**0**]** dSUM**[**0**:**W**];** // Массив для хранения промежуточных результатов суммы

**reg** **[** W**-**1**:**0**]** A **[**0**:**W**];** // Массив для хранения промежуточных значений операнда A

**reg** **[** W**-**1**:**0**]** B **[**0**:**W**];** // Массив для хранения промежуточных значений операнда B

// Начальные значения для массивов и результата

**initial** **begin**

**for** **(int** i **=** 0**;** i **<=** W**;** i**++)** **begin**

dSUM**[**i**]** **<=** 1'b0**;**

A**[**i**]** **<=** 1'b0**;**

B**[**i**]** **<=** 1'b0**;**

**end**

a\_mult\_b **<=** 1'b0**;**

**end**

// На каждом положительном фронте тактового сигнала, присваиваем значения dA и dB соответственно в A[0] и B[0]

**always** **@(posedge** clk**)** **begin**

A**[**0**]** **<=** dA**;**

B**[**0**]** **<=** dB**;**

**end**

**genvar** i**;**

**generate**

// Используем генерацию кода для создания параллельных блоков (генераторов) для каждого бита входного операнда

**for** **(**i **=** 0**;** i **<** W**;** i**++)** **begin** **:** generator

// На каждом положительном фронте тактового сигнала, обновляем промежуточные значения dSUM, A и B

**always** **@(posedge** clk**)** **begin**

dSUM**[**i**+**1**]** **<=** **(**dSUM**[**i**]** **<<** 1'b1**)** **+** **(**B**[**i**][**W**-**i**-**1**]** **?** A**[**i**]** **:** 1'b0**);**

A**[**i**+**1**]** **<=** A**[**i**];**

B**[**i**+**1**]** **<=** B**[**i**];**

**end**

**end**

**endgenerate**

// На каждом положительном фронте тактового сигнала, присваиваем значение dSUM[W] регистру a\_mult\_b

**always** **@(posedge** clk**)** a\_mult\_b **<=** dSUM**[**W**];**

**endmodule**

Эта реализация отличается от прошлой тем, что следующее действие выполняется каждый такт, вследствие чего необходимо хранить используемые для суммирования переменные для каждого такта.

Выполним моделирование разработанного модуля. Для этого разработаем следующий тестовый модуль, который перебирает все варианты входных параметров:

`timescale 1ns **/** 1ns

`include "mult\_LP\_V1.sv"

**module** mult\_LP\_V1\_tb**;**

**parameter** PERIOD **=** 10**;**

**parameter** W **=** 8**;**

**reg** clk **=** 0**;**

**reg** **[** W**-**1**:**0**]** dA **[**0**:**W**+**1**];**

**reg** **[** W**-**1**:**0**]** dB **[**0**:**W**+**1**];**

**wire** **[**2**\***W**-**1**:**0**]** a\_mult\_b**;**

**initial** **begin**

**forever** **#(**PERIOD **/** 2**)** clk **=** **~**clk**;**

**end**

mult\_LP\_V1 **#(**

**.**W**(**W**)**

**)** u\_mult\_LP\_V1 **(**

**.**clk**(**clk**),**

**.**dA **(**dA**[**0**][**W**-**1**:**0**]),**

**.**dB **(**dB**[**0**][**W**-**1**:**0**]),**

**.**a\_mult\_b**(**a\_mult\_b**[**2**\***W**-**1**:**0**])**

**);**

**initial** **begin**

**for** **(int** i **=** 0**;** i **<** $pow**(**2**,** W**);** i**++)**

**for** **(int** j **=** 0**;** j **<** $pow**(**2**,** W**);** j**++)** **begin**

**for** **(int** k **=** W **+** 1**;** k **>** 0**;** k**--)** **begin**

dA**[**k**]** **=** dA**[**k**-**1**];**

dB**[**k**]** **=** dB**[**k**-**1**];**

**end**

dA**[**0**]** **=** i**;**

dB**[**0**]** **=** j**;**

**#(**PERIOD**);**

**if** **(**dA**[**W**+**1**]** **\*** dB**[**W**+**1**]** **!=** a\_mult\_b**)** **begin**

$display**(**"Incorrect result: %0d \* %0d = %0d\nExpected: %0d"**,** dA**[**W**+**1**],** dB**[**W**+**1**],** a\_mult\_b**,**

dA**[**W**+**1**]** **\*** dB**[**W**+**1**]);**

$stop**;**

**end**

**end**

**for** **(int** i **=** 0**;** i **<=** W**;** i**++)** **begin**

**for** **(int** k **=** W **+** 1**;** k **>** 0**;** k**--)** **begin**

dA**[**k**]** **=** dA**[**k**-**1**];**

dB**[**k**]** **=** dB**[**k**-**1**];**

**end**

dA**[**0**]** **=** 0**;**

dB**[**0**]** **=** 0**;**

**#(**PERIOD**);**

**if** **(**dA**[**W**+**1**]** **\*** dB**[**W**+**1**]** **!=** a\_mult\_b**)** **begin**

$display**(**"Incorrect result: %0d \* %0d = %0d\nExpected: %0d"**,** dA**[**W**+**1**],** dB**[**W**+**1**],** a\_mult\_b**,**

dA**[**W**+**1**]** **\*** dB**[**W**+**1**]);**

$stop**;**

**end**

**end**

$display**(**"All tests passed!"**);**

$stop**;**

**end**

**endmodule**

Этот модуль отличается тем, что задержка между получением данных и результатом составляет больше, чем 1 такт. Выполним что-то похожее на то, что мы реализовали в самом модуле и будем хранить значение подаваемых данных каждый такт и подавать следующие, сравнивая их с задержкой в W тактов.

Это породило проблему с тем, что после перебора всех вариантов результат ещё необходимо дождаться, это было реализовано вторым циклом.

Запустим моделирование в среде ModelSim с W = 8:

Аналогично предыдущему результату на Wave, который был описан выше, Wave имеет большое кол-во значений, что затрудняет считывание полученных данных:

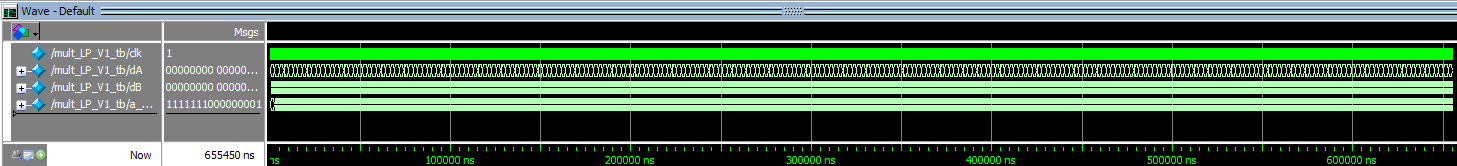


Рис. 2.12 – Wave для mult\_LP\_V1\_tb

Однако сообщение в консоли свидетельствует о том, что все тесты выполненные корректно:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

Рис. 2.13 – Transcipt для mult\_LP\_V1\_tb

Выполним компиляцию модуля mult\_LP\_V1 в среде Quartus, с параметром W = 4:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рис. 2.14 – Результат компиляции mult\_LP\_V1 с W = 4

Также посмотрим на максимальную частоту устройства:

Изображение выглядит как текст, линия, Шрифт, программное обеспечение

Автоматически созданное описание

Рис. 2.15 – Максимальная частота устройства mult\_LP\_V1 с W = 4

Повторим скриншоты с W = 8 и W = 16:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рис. 2.16 – Результат компиляции mult\_LP\_V1 с W = 8

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

Рис. 2.17 – Максимальная частота устройства mult\_LP\_V1 с W = 8

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рис. 2.18 – Результат компиляции mult\_LP\_V1 с W = 16

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рис. 2.19 – Максимальная частота устройства mult\_LP\_V1 с W = 16

Как можно заметить, чем больше разрядность, чем больше затрачиваемые ресурсы, однако вследствие конвейеризации максимальная частота устройства практически не уменьшается.

Также стоит рассмотреть RTL Viewer разработанного устройства. Для удобства анализа поставим W = 4:

Изображение выглядит как диаграмма, линия, План, снимок экрана

Автоматически созданное описание

Рис. 2.20 – RTL Viwer для mult\_LP\_V1

Как видно, каждая итерация из блока MUX и блока ADD. MUX используются для того, чтоб определить, делать ли сумму или нет, после чего идет ряд регистров для сохранения промежуточных результатов и данных для умножения.

Составим таблицу с данными по затрачиваемым ресурсам и частоте устройств:

Табл. 2.21 – Аппаратные затраты и частоты устройств

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Аппаратные затраты | | Максимальная тактовая частота |
| Части lab\_5s | Разрядность | Logic Elements | Registers | (МГц) |
| Часть 1 | 4 | 34 | 16 | 171.91 |
| 8 | 152 | 32 | 78.51 |
| 16 | 624 | 64 | 36.48 |
| Часть 2 | 4 | 59 | 59 | 343.41 |
| 8 | 220 | 215 | 281.06 |
| 16 | 830 | 815 | 195.77 |

Как видим, конвейеризация действительно значительно повышает частоту работы устройства, однако, как видно, сильно повышает затрачиваемые ресурсы т. к. на каждом этапе появляются регистры, в которых хранятся данные. Так же повышаются и логические затраты т. к., видимо, компилятор не смог выполнить некоторые оптимизации, которые были доступны в не конвейерном режиме.

# Вывод:

В ходе лабораторной работы было создано устройство для умножения чисел различной разрядности, ориентированное на схему "старшие разряды вперед со сдвигом суммы". Процесс разработки включал тестирование с применением модулей, подтвердивших правильность предложенной схемы. Затем мы реализовали конвейеризацию устройства, что, как оказалось на практике, значительно увеличивает тактовую частоту работы устройства. Вместе с тем, следует отметить, что этот подход сопряжен с высоким потреблением ресурсов.

Разработанное устройство, несмотря на увеличение потребления ресурсов, обладает высокой эффективностью в плане тактовой частоты. Этот опыт подтверждает, что выбор схемы "старшие разряды вперед со сдвигом суммы" обоснован и имеет практическое применение в области разработки устройств умножения.

Использование языка Verilog значительно упрощает процесс разработки сложных устройств, предоставляя возможность использования блочной диаграммы и параметров для быстрого внесения изменений. Кроме того, инструмент ModelSim дает возможность проводить более глубокие и комплексные тестирования устройства, что значительно ускоряет весь процесс разработки. Современные методы и инструменты играют ключевую роль в повышении эффективности и ускорении разработки сложных электронных устройств.