Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных средств

Дисциплина: Проектирование цифровых систем на языках описания аппаратуры

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

СИСТОЛИЧЕСКИЙ ПРОЦЕССОР УМНОЖЕНИЯ МАТРИЦ 8 НА 8

БГУИР КП 1-40 02 02 049 ПЗ

|  |  |
| --- | --- |
|  | Студент: гр. 850702 Турко В. Д. |
|  | Руководитель: ассистент Рыбенков Е.В |

Минск 2020

Содержание

Введение 3

1 Теоретические сведения 4

1.1 Систолический процессор 4

1.2 Отображение алгоритма умножения на матричный массив 4

2 Разработка VHDL-описания 8

2.1 Используемые типы 8

2.2 VHDL-описание процессорных элементов 8

2.3 VHDL-описание процессора умножения 9

3 Синтез и моделирование 12

3.1 Введение 12

3.2 Синтез принципиальной схемы 12

3.3 Моделирование 13

Заключение 15

Список использованных источников 16

ПРИЛОЖЕНИЕ А VHDL-описание типа матрицы и констант 17

ПРИЛОЖЕНИЕ Б VHDL-описание процессорного элемента 18

ПРИЛОЖЕНИЕ В VHDL-описание процессора умножения матриц 19

ПРИЛОЖЕНИЕ Г Код тестовой программы 22

ПРИЛОЖЕНИЕ Д Схема принципиальная 23

# Введение

Возрастающие требования к скорости и производительности решения современных задач параллельных вычислений требуют развития организации параллельной работы в вычислительных машинах, одним из направлений которого является репликация однотипных устройств с регулярной топологии связи, т.е. создание матричных систем. Узлы матричной системы выполняют одну и ту же операцию над всеми элементами массива данных. В ходе освоения матричных систем были предложены идеи организации вычислений на подобных структурах, которые основаны на конвейерном представлении алгоритмов решения задач, — систолических вычислениях ([1], с. 5).

Систолические структуры эффективны при выполнении матричных вычислений, сортировке данных, а также задач обработки сигналов, обработки изображений и др., решения которых часто требуются в реальном времени.

Каждый систолический процессор решает одну конкретную задачу или класс задач. Например, в большинстве задач обработки сигналов и изображений преобладают методы преобразований, фильтрации и базовые методы линейной алгебры. Поэтому данное курсовое проектирования посвящено умножению матриц, а его темой является «Систолический процессор умножения матриц 8 на 8».

Целью данной курсовой работы является разработка систолического процессора умножения матриц 8 на 8. В соответствии с поставленной целью необходимо решить разработать VHDL-описание умножителя и решить задачу синтеза принципиальной схемы.

В результате выполнения данного курсового проектирования была спроектирована схема систолического процессора умножения матриц 8 на 8 и проведено моделирование её работы, результатом которого являются временные диаграммы для различных наборов данных.

# Теоретические сведения

## Систолический процессор

Систолические процессоры хорошо приспособлены для реализации на СБИС. Особенно пригодны они для специального класса вычислительных алгоритмов с регулярным, локализованным потоком данных, управляемым глобальной потактовой синхронизацией.

Систолическая система – это сеть процессоров, которые ритмично вычисляют и передают данные по системе ([1], с. 158).

Систолический процессор обладает свойствами модульности, регулярности, локальности связей, высокой степенью конвейеризации и максимально синхронной мультиобработки ([1], с. 159). Систолический массив также является тиражируемым, что проявляется в возможности неограниченно расширять его.

Так несколько процессоров или процессорных элементов (далее – ПЭ), выполняющий операцию скалярного произведения, могут быть объединены с помощью локальных связей для выполнения различных операций, например, умножения матриц.

## Отображение алгоритма умножения на матричный массив

Простой метод определения работоспособности матричной структуры, соответствующей локально рекурсивному алгоритму, которым является умножение матриц, заключается в назначении ПЭ для каждого узла графа зависимостей. Граф зависимостей можно рассматривать в более сжатом представлении: в виде графа потока сигналов (ГПС). ГПС более близок к аппаратному уровню и определяет тип матричного устройство.

Полное описание ГПС состоит из функциональной и структурной частей. Функциональное описание представляет поведение в узле, а структурные связи – между узлами.

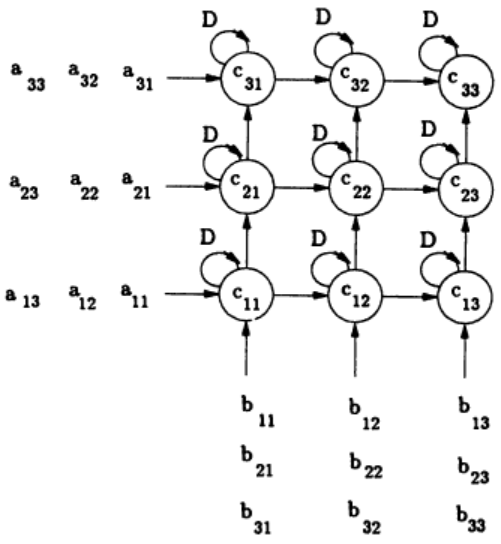
Умножение матриц означает вычисление элементов матрицы С в виде:

Из этого выражения видно, что вычисления для каждого элемента матрицы С могут быть выполнены одновременно, так как между ними нет никаких зависимостей. Граф зависимости для алгоритма умножения матриц представлен на рисунке 1.1.

|  |  |
| --- | --- |
| Структурный уровень | Функциональный уровень |
|  |  |

#### ГЗ перемножения матриц

После проекции ГЗ в направлении [0 0 1] получим ГПС, представленный на рисунке 1.2. Этот ГПС является пространственно-локализованным, но не временно-локализованным; т.е. данные распространяются без задержек, поэтому необходимо произвести ресинхронизацию.



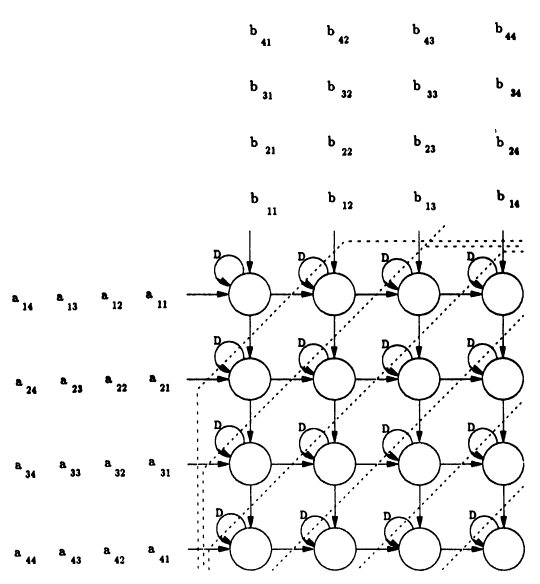
#### ГПС для перемножения матриц

Для дальнейшей разработки систолического массива необходимо преобразование ГПС в систолический массив, иначе говоря, провести систолизацию. Данная процедура включает в себя выбор основных функциональных модулей, применение правил ресинхронизации и объединение задержки с функциональным модулем.

В качестве функционального модуля выберем «скалярное умножение», поведение которого описано на рисунке 1.1. Также будем считать, что данная операция выполняет за 1 такт, что и будет задержкой модуля.

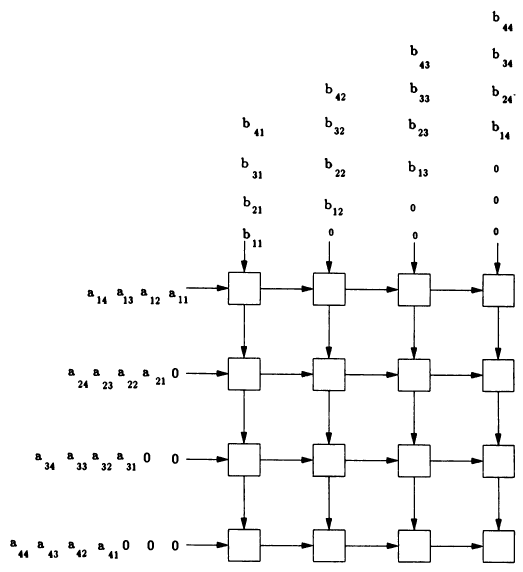
Ресинхронизацию регулярного ГПС можно осуществить с использованием регулярных сечений и правил ресинхронизации, которые описаны в книге С. Куна «Матричные процессоры на СБИС» (с.218-219).

Как показано на рисунке 1.2 ГПС обычного умножения матриц допускает одновременное распространение столбцов А и строк В по квадратному массиву с частичной суммой внешних произведений, возвращаемых по петле с задержкой. Применим правило перемещения задержки к сечениям, показанным на рисунке 1.3.



#### Сечения ГПС для умножения матриц

В соответствии с этим правилом ввод различных столбцов В и строк А должен быть до поступления в массив отрегулирован определённым количеством задержек. Результат подсчета сечений говорит о том, что матрицы А и В должны быть скошены, как показано на рисунке 1.4.



#### Систолический массив для умножения матриц

# Разработка VHDL-описания

## Используемые типы

Для работы с матрицами необходимо объявить в пакете соответствующий тип. В качестве элемента матрицы был использован бит-вектор, а в качестве матрицы – двумерный массив. Для удобства демонстрации элемент матрицы имеет 4 разряда, а сами матрицы имеют размер 4 на 4. Как будет показано далее, количество разрядов элемента матрицы и размер матриц можно менять, не внося значительных изменений в код.

Т.к. входные данные состоят из 4-х разрядных элементов, элементы выходных данных могут иметь до 8 ненулевых разрядов. И поскольку VHDL не позволяет использовать бит-вектор произвольного размера при объявлении типа, для входных и выходных данных необходимо объявить 2 типа.

Полный описание пакетов с константами и объявлениями типов приведен в приложении А.

PACKAGE matrix\_package IS

    TYPE matrix IS ARRAY (POSITIVE RANGE <>, POSITIVE RANGE <>)

        OF STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0);

    TYPE result\_matrix IS ARRAY (POSITIVE RANGE <>, POSITIVE RANGE <>)

        OF STD\_LOGIC\_VECTOR(RESULT\_SIZE DOWNTO 0);

END PACKAGE matrix\_package;

*EL\_SIZE* и *RESULT\_SIZE* – константы, задающие количество зарядов данных, значения которых – 4 и 8 соответственно.

## VHDL-описание процессорных элементов

Каждый процессорный элемент имеет порт асинхронного сброса *R*, порт синхронизации *clk*, входные порты *a* и *b* для приема данных, выходные порты *a\_out* и *b\_out* для продвижения данных по процессору, а также порт *С*, на который подается результат вычисления частичной суммы элемента выходных данных. Далее приведено VHDL-описание интерфейса процессорного элемента. Полное описание ПЭ содержится в приложении В.

Вычисление данных и дальнейшее их распространение по процессору происходит каждый такт синхронизации, что можно трактовать как задержку величиной в один такт.

ENTITY MatrixProc IS

    GENERIC (use\_a\_out : BOOLEAN := TRUE; use\_b\_out : BOOLEAN := TRUE);

    PORT (

        R, clk : IN STD\_LOGIC;

        a : IN STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

        b : IN STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

        a\_out : OUT STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

        b\_out : OUT STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

        c : INOUT STD\_LOGIC\_VECTOR(RESULT\_SIZE DOWNTO 0) := (OTHERS => '0'));

END MatrixProc;

*Generic*-поля *use\_a\_out* и *use\_b\_out* позволяют указывать, в каком направлении данный ПЭ должен распространять данные. Так, например, нижний правый элемент на рисунке 1.4 является крайним и не распространяет данные, для него данные поля имеют значение false. Часть VHDL-описания, реализующее необходимое поведение, приведена ниже.

IF rising\_edge(clk) THEN

    IF (use\_A\_out) THEN a\_out <= a; END IF;

    IF (use\_b\_out) THEN b\_out <= b; END IF;

c <= STD\_LOGIC\_VECTOR(unsigned(C) + unsigned(A) \* unsigned(B));

END IF;

В данном описании вычисление частичной суммы элемента матрицы происходит на уровне слов. Также систолические массивы можно рассматривать на разрядном уровне, которые предполагает поразрядную обработку в каждом ПЭ. Такие системы позволяют увеличить скорость конвейерной обработки, однако в данной работе в целях упрощения систолический массив процессора умножения матриц рассматривается на уровне слов.

## VHDL-описание процессора умножения

Интерфейс процессора умножения матриц содержит порт асинхронного сброса *R*, порт синхронизации *clk*, входные порты *a* и *b* для приема данных, выходной порт с для вывода результата вычислений. Значение логической 1 сигнала порта *ready* говорит о завершении вычислений, а значение логического 0 – об обратном.

*Generic*-поле *N* позволяет регулировать размер систолического массива в соответствии с размером обрабатываемых матриц. В данной работе для удобства отображения были выбраны матрицы размером 4 на 4.

ENTITY MatrixMulGen IS

    GENERIC (N : POSITIVE := M\_SIZE);

    PORT (

        R, clk : IN STD\_LOGIC;

        a : IN MATRIX(1 TO N, 1 TO N);

        b : IN MATRIX(1 TO N, 1 TO N);

        ready : INOUT STD\_LOGIC;

        c : OUT RESULT\_MATRIX(1 TO N, 1 TO N));

END MatrixMulGen;

Входные данные (матрицы) подаются в процессор в виде, представленном на рисунке 2.1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a11 | a12 | a13 | a14 | × | b11 | b12 | b13 | b14 | = | c11 | c12 | c13 | c14 |
| a21 | a22 | a23 | a24 | b21 | b22 | b23 | b24 | c21 | c22 | c23 | c24 |
| a31 | a32 | a33 | a34 | b31 | b32 | b33 | b34 | c31 | c32 | c33 | c34 |
| a41 | a42 | a43 | a44 | b41 | b42 | b43 | b44 | c41 | c42 | c43 | c44 |

#### Представление матриц

Перед подачей непосредственно в систолический массив, матрицы *А* и *В* должны быть представлены в виде, показанном на рисунке 1.4, т.е. отражены относительно вертикальной и горизонтальной осей соответственно. Также необходимо скосить матрицы с помощью задержек. Представленная ниже часть описания процессора регулирует подачу данных в массив, подавая значение из матрицы входных данных или нулевой вектор, в зависимости от условий.

    GEN\_SHIFT\_FIRST : FOR I IN N DOWNTO 1 GENERATE

        PROCESS (counter)

            VARIABLE tmp : INTEGER;

        BEGIN

            tmp := to\_integer(unsigned(counter)) - I + 1;

            IF (tmp > 0 AND tmp <= N) THEN

                A\_inner(I, 1) <= A(I, tmp);

                B\_inner(1, I) <= B(tmp, I);

            ELSE

                A\_inner(I, 1) <= (OTHERS => '0');

                B\_inner(1, I) <= (OTHERS => '0');

            END IF;

        END PROCESS;

    END GENERATE GEN\_SHIFT\_FIRST;

Компоненты процессорных элементов вставляются в массив с помощью операторов *generate* и *port map*, что позволяет не описывать каждый процессорный элемент, а также изменять размер систолического массива в соответствии с размерами входных матриц с помощью изменения значения *generic*-поля *N*.

GEN\_PROC\_ROWS : FOR I IN 1 TO N - 1 GENERATE

    GEN\_PROC\_COLUMNS : FOR J IN 1 TO N - 1 GENERATE

        pI : MatrixProc PORT MAP(

            R => R,

            clk => clk,

            a => A\_inner(I, J),

            b => B\_inner(I, J),

            a\_out => A\_inner(I, J + 1),

            b\_out => B\_inner(I + 1, J),

            c => c\_result(I, J));

    END GENERATE GEN\_PROC\_COLUMNS;

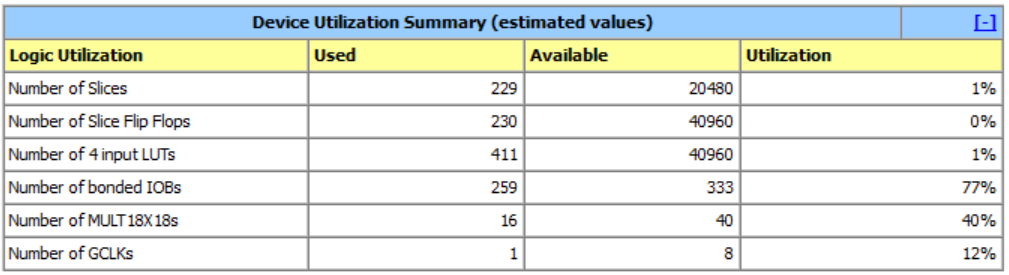
Полное VHDL-описание процессора умножения матриц представлено в приложении В.

# Синтез и моделирование

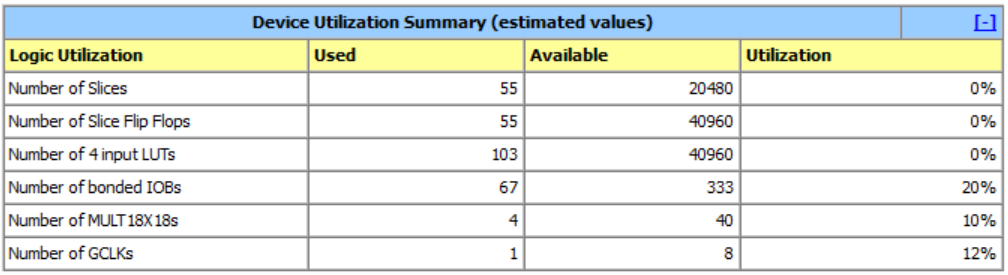
Синтез производился в среде проектирования Xilinx ISE на базе устройства XC3S2000 из семейства кристаллов Spartan 3. Для моделирования был использован встроенный в среду проектирования симулятор ISim.

## Синтез принципиальной схемы

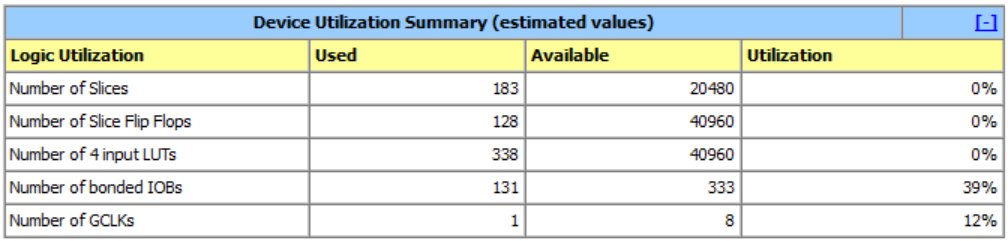
Результаты синтеза принципиальных схем в Xilinx ISE для различных комбинаций размеров матриц и разрядности элементов матриц показаны на рисунках 3.1-3.4. По результатам синтеза схем можно сделать вывод, что увеличение размера обрабатываемых матриц вносит больший вклад в увеличение аппаратных затрат, чем увеличение разрядности данных. Причем систолический процессор умножения матриц 4 на 4 для 8-ми разрядных чисел на выбранном устройстве реализовать невозможно, т.к. процент использованных ресурсов устройства XC3S2000 превысил 100%.



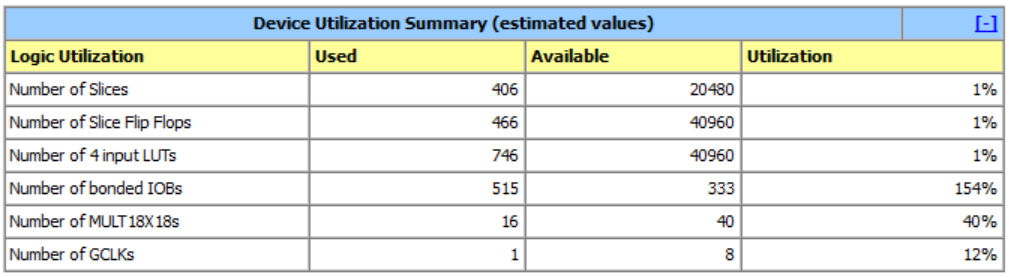
#### Результат синтеза умножителя матриц 4-х разрядных чисел 4 на 4.



#### Результат синтеза умножителя матриц 4-х разрядных чисел 2 на 2.



#### Результат синтеза умножителя матриц 2-х разрядных чисел 4 на 4.



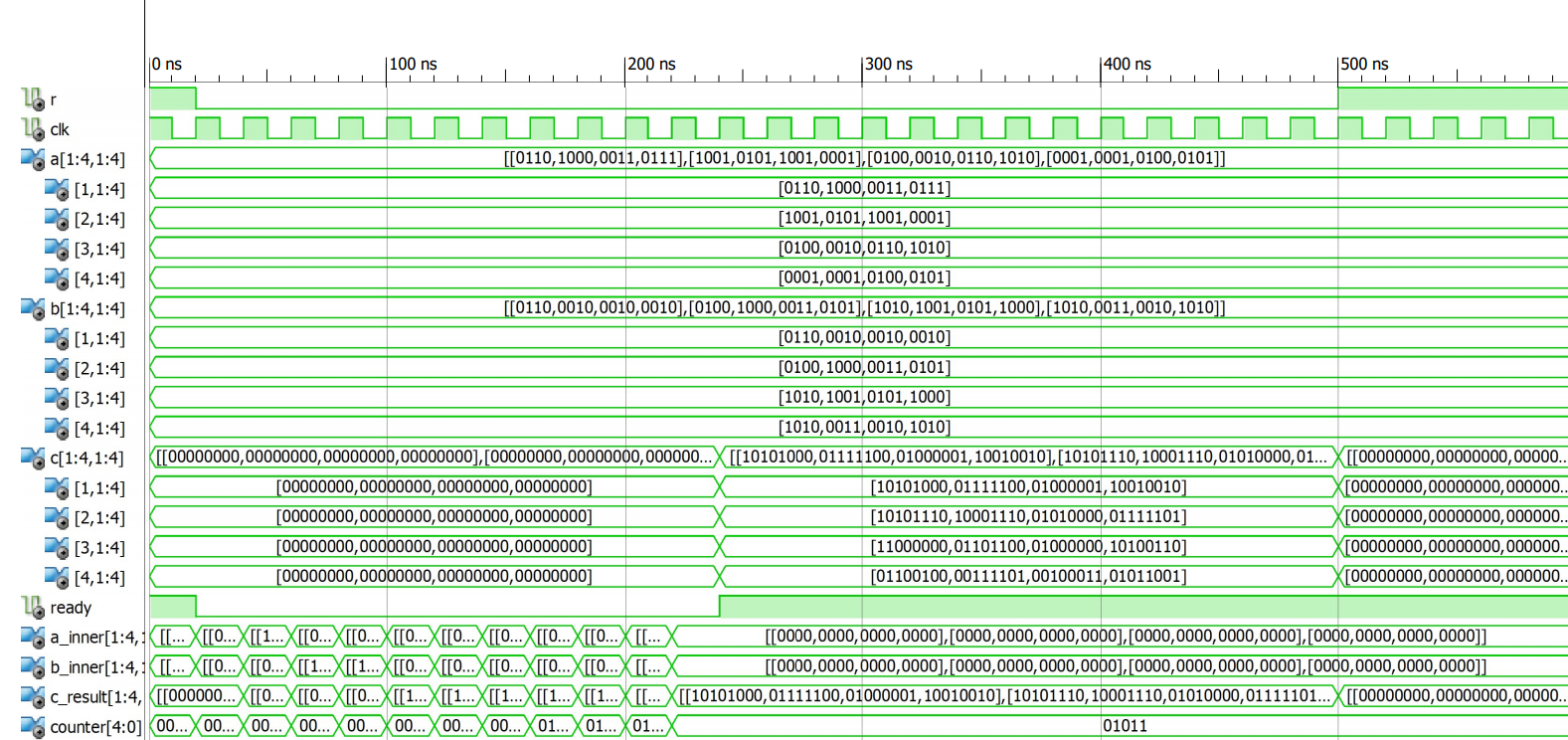
#### Результат синтеза умножителя матриц 8-х разрядных чисел 4 на 4.

## Моделирование

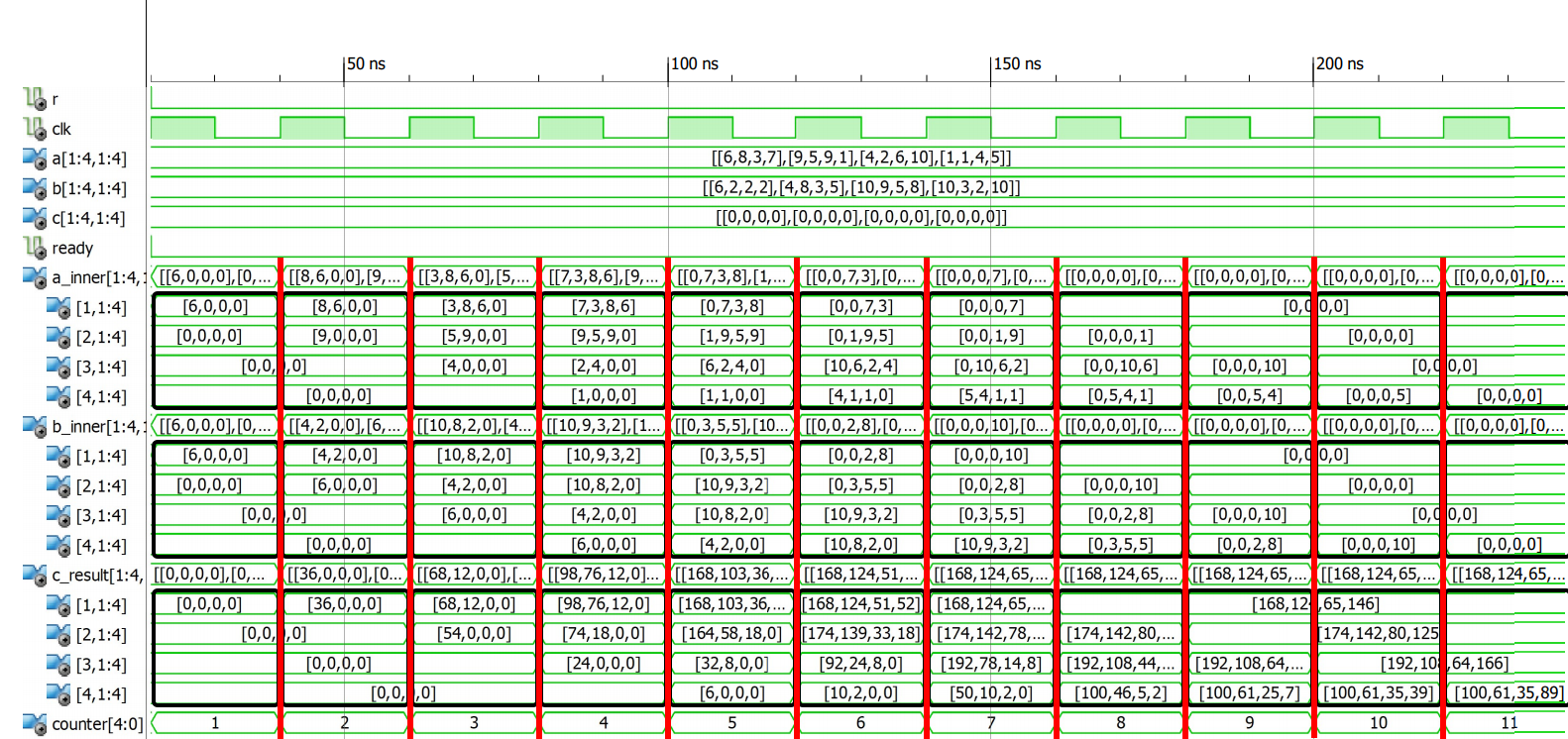
Для моделирования работы процессора умножения матриц был использован VHDL test bench, пример которого приведен в приложении Д. Показанные далее временные диаграммы (рис. 3.5-3.6) были получены в результате моделирования работы процессора умножения с матрицами, содержащимися в VHDL-описании в приложении Д.

На рисунке 3.5 показана общая временная диаграмма работы процессора в промежутке времени от 0 нс до 600 нс, на которой можно увидеть входные и выходные данные, а также менее подробно – внутренние сигналы процессора.

На рисунке 3.6 более подробное показаны внутренние сигналы систолического массива, а именно показан участок временной диаграммы в промежутке от 20 нс до 240 нс. Для удобства наблюдения система счисления для отображения была изменена с *binary* на *unsigned decimal.* Внутренние сигналы систолического массива удобно рассматривать в виде кадров – описания действий в конкретный момент времени. Так на временной диаграмме можно увидеть элементы данных, содержащиеся в каждом процессорном элементе в различные моменты времени.



#### Временная диаграмма работы систолического процессора умножения матриц 4 на 4.



#### Кадры на временной диаграмме для систолического массива умножения матриц 4 на 4.

Как видно из временной диаграммы 3.6, необходимое количество тактов длины T для умножения квадратных матриц размера на равно , а общее время выполнения вычислений – .

# Заключение

Основной принцип систолической разработки заключается в достижении массового параллелизма при минимальных тратах на связь. Свойства модульности, регулярности и локальности связей делают систолические структуры доступными для реализации на СБИС. Однако их применение ограничено специальным классом алгоритмов, которые используют регулярные и локальные структуры данных, которые естественным образом соответствуют систолическим структурам. Примером такого алгоритма является умножение матриц, поэтому реализованный в данной работе процессор умножения матриц является специализированной системой, однако систолические структуры могут иметь некоторую степень гибкости.

При разработке реальных систем придется столкнуться с различными техническими проблемами, к которым относятся реализация схемы синхроимпульсов и реализация интерфейса между основной машиной и систолическим процессором. Как было показано в разделе 3.2 данной работы, увеличение размера обрабатываемых матриц или увеличение количества разрядов их элементов влечет за собой стремительное увеличение использования ресурсов устройства. Ведь для обработки матриц размера   
 на , содержащих *M*-разрядные элементы, необходимо процессорных элементов, а также *M*-разрядных портов ввода и 2M-разрядных портов вывода.

# Список использованных источников

###### Кун, С. Матричные процессоры на СБИС / С. Кун; Перевод с англ. Ю. Г. Дадаева и др.; Под ред. Ю. Г. Дадаева. - М. : Мир, 1991. - 672 с.

VHDL-описание типа матрицы и констант

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.ALL;

USE IEEE.numeric\_std.ALL;

PACKAGE utils\_package IS

    CONSTANT clk\_period : TIME := 20 ns;

    CONSTANT EL\_BITS : INTEGER := 4;

    CONSTANT EL\_SIZE : INTEGER := EL\_BITS - 1;

    CONSTANT RESULT\_SIZE : INTEGER := EL\_BITS \* 2 - 1;

    CONSTANT M\_SIZE : INTEGER := 4;

END PACKAGE utils\_package;

PACKAGE matrix\_package IS

    TYPE matrix IS ARRAY (POSITIVE RANGE <>, POSITIVE RANGE <>)

        OF STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0);

    TYPE result\_matrix IS ARRAY (POSITIVE RANGE <>, POSITIVE RANGE <>)

        OF STD\_LOGIC\_VECTOR(RESULT\_SIZE DOWNTO 0);

END PACKAGE matrix\_package;

VHDL-описание процессорного элемента

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.numeric\_std.ALL;

LIBRARY work;

USE work.utils\_package.ALL;

PACKAGE processors\_package IS

    COMPONENT MatrixProc IS

        GENERIC (use\_a\_out : BOOLEAN := TRUE; use\_b\_out : BOOLEAN := TRUE);

        PORT (

            R, clk : IN STD\_LOGIC;

            a : IN STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

            b : IN STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

            a\_out : OUT STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

            b\_out : OUT STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

            c : INOUT STD\_LOGIC\_VECTOR(RESULT\_SIZE DOWNTO 0) := (OTHERS => '0'));

    END COMPONENT;

END PACKAGE processors\_package;

ENTITY MatrixProc IS

    GENERIC (use\_a\_out : BOOLEAN := TRUE; use\_b\_out : BOOLEAN := TRUE);

    PORT (

        R, clk : IN STD\_LOGIC;

        a : IN STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

        b : IN STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

        a\_out : OUT STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

        b\_out : OUT STD\_LOGIC\_VECTOR(EL\_SIZE DOWNTO 0) := (OTHERS => '0');

        c : INOUT STD\_LOGIC\_VECTOR(RESULT\_SIZE DOWNTO 0) := (OTHERS => '0'));

END MatrixProc;

ARCHITECTURE MatrixProcArch OF MatrixProc IS

BEGIN

    PROCESS (R, clk)

    BEGIN

        IF (R = '1') THEN

            c <= (OTHERS => '0');

        ELSE

            IF rising\_edge(clk) THEN

                IF (use\_A\_out) THEN a\_out <= a; END IF;

                IF (use\_b\_out) THEN b\_out <= b; END IF;

                c <= STD\_LOGIC\_VECTOR(unsigned(C) + unsigned(A) \* unsigned(B));

            END IF;

        END IF;

    END PROCESS;

END ARCHITECTURE MatrixProcArch;

VHDL-описание процессора умножения матриц

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.numeric\_std.ALL;

USE ieee.math\_real.ALL;

LIBRARY STD;

USE std.textio.ALL;

LIBRARY work;

USE work.utils\_package.ALL;

USE work.matrix\_package.ALL;

USE work.processors\_package.ALL;

ENTITY MatrixMulGen IS

    GENERIC (N : POSITIVE := M\_SIZE);

    PORT (

        R, clk : IN STD\_LOGIC;

        a : IN MATRIX(1 TO N, 1 TO N);

        b : IN MATRIX(1 TO N, 1 TO N);

        ready : INOUT STD\_LOGIC;

        c : OUT RESULT\_MATRIX(1 TO N, 1 TO N));

END MatrixMulGen;

ARCHITECTURE MatrixMulGenArch OF MatrixMulGen IS

    SIGNAL a\_inner : MATRIX(1 TO N, 1 TO N);

    SIGNAL b\_inner : MATRIX(1 TO N, 1 TO N);

    SIGNAL c\_result : RESULT\_MATRIX(1 TO N, 1 TO N);

    SIGNAL counter : STD\_LOGIC\_VECTOR(

INTEGER(ceil(log2(real(3 \* N - 1)))) DOWNTO 0) := (OTHERS => '0');

BEGIN

    ready\_observer : PROCESS (R, clk, counter)

    BEGIN

        IF (R = '1') THEN

            ready <= '1';

        ELSE

            IF rising\_edge(clk) THEN

                IF (unsigned(counter) = 3 \* N - 1) THEN

                    ready <= '1';

                ELSE

                    counter <= STD\_LOGIC\_VECTOR(unsigned(counter) + 1);

                    ready <= '0';

                END IF;

            END IF;

        END IF;

    END PROCESS;

    GEN\_SHIFT\_FIRST : FOR I IN N DOWNTO 1 GENERATE

        PROCESS (counter)

            VARIABLE tmp : INTEGER;

        BEGIN

            tmp := to\_integer(unsigned(counter)) - I + 1;

            IF (tmp > 0 AND tmp <= N) THEN

                A\_inner(I, 1) <= A(I, tmp);

                B\_inner(1, I) <= B(tmp, I);

            ELSE

                A\_inner(I, 1) <= (OTHERS => '0');

                B\_inner(1, I) <= (OTHERS => '0');

            END IF;

        END PROCESS;

    END GENERATE GEN\_SHIFT\_FIRST;

    GEN\_RESULT\_ROWS : FOR I IN N DOWNTO 1 GENERATE

        GEN\_RESULT\_COLUMNS : FOR J IN N DOWNTO 1 GENERATE

            PROCESS (ready, c\_result)

            BEGIN

                IF (ready = '1') THEN

                    c(I, J) <= c\_result(I, J);

                ELSE

                    c(I, J) <= (OTHERS => '0');

                END IF;

            END PROCESS;

        END GENERATE GEN\_RESULT\_COLUMNS;

    END GENERATE GEN\_RESULT\_ROWS;

    GEN\_PROC\_ROWS : FOR I IN 1 TO N - 1 GENERATE

        GEN\_PROC\_COLUMNS : FOR J IN 1 TO N - 1 GENERATE

            pI : MatrixProc PORT MAP(

                R => R,

                clk => clk,

                a => A\_inner(I, J),

                b => B\_inner(I, J),

                a\_out => A\_inner(I, J + 1),

                b\_out => B\_inner(I + 1, J),

                c => c\_result(I, J));

        END GENERATE GEN\_PROC\_COLUMNS;

        pA : MatrixProc GENERIC MAP(use\_a\_out => FALSE)

        PORT MAP(

            R => R,

            clk => clk,

            a => A\_inner(I, N),

            b => B\_inner(I, N),

            b\_out => B\_inner(I + 1, N),

            c => c\_result(I, N));

        pB : MatrixProc GENERIC MAP(use\_B\_out => FALSE)

        PORT MAP(

            R => R,

            clk => clk,

            a => A\_inner(N, I),

            b => B\_inner(N, I),

            a\_out => A\_inner(N, I + 1),

            c => c\_result(N, I));

    END GENERATE GEN\_PROC\_ROWS;

    pL : MatrixProc GENERIC MAP(use\_A\_out => FALSE, use\_B\_out => FALSE)

    PORT MAP(

        R => R,

        clk => clk,

        a => A\_inner(N, N),

        b => B\_inner(N, N),

        c => c\_result(N, N));

END ARCHITECTURE MatrixMulGenArch;

Код тестовой программы

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE work.matrix\_package.ALL;

USE work.utils\_package.ALL;

ENTITY MatrixTest IS

END MatrixTest;

ARCHITECTURE MatrixTestArch OF MatrixTest IS

    COMPONENT MatrixMulGen IS

        GENERIC (N : POSITIVE := M\_SIZE);

        PORT (

            R, clk : IN STD\_LOGIC;

            a : IN MATRIX(1 TO N, 1 TO N);

            b : IN MATRIX(1 TO N, 1 TO N);

            ready : INOUT STD\_LOGIC;

            c : OUT RESULT\_MATRIX(1 TO N, 1 TO N));

    END COMPONENT;

    SIGNAL R : STD\_LOGIC := '1';

    SIGNAL clk : STD\_LOGIC := '1';

    SIGNAL a : MATRIX(1 TO M\_SIZE, 1 TO M\_SIZE) := (

        ("0110", "1000", "0011", "0111"),

        ("1001", "0101", "1001", "0001"),

        ("0100", "0010", "0110", "1010"),

        ("0001", "0001", "0100", "0101"));

    SIGNAL b : MATRIX(1 TO M\_SIZE, 1 TO M\_SIZE) := (

        ("0110", "0010", "0010", "0010"),

        ("0100", "1000", "0011", "0101"),

        ("1010", "1001", "0101", "1000"),

        ("1010", "0011", "0010", "1010"));

    SIGNAL c : RESULT\_MATRIX(1 TO M\_SIZE, 1 TO M\_SIZE);

    SIGNAL ready : STD\_LOGIC := '0';

BEGIN

    portm : MatrixMulGen PORT MAP(R, clk, a, b, ready, c);

    clk\_process : PROCESS

    BEGIN

        clk <= not clk after clk\_period / 2;

    END PROCESS;

    R\_process : PROCESS

    BEGIN

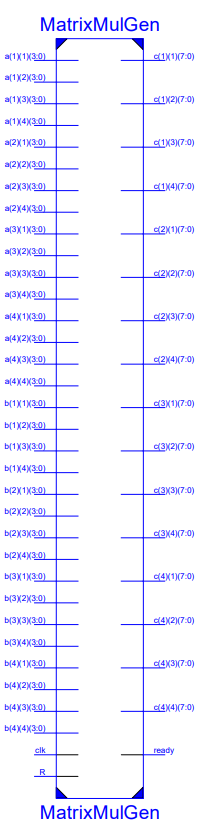
R <= not R after clk\_period;

        WAIT;

    END PROCESS;

END ARCHITECTURE MatrixTestArch;

Схема принципиальная

****