

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

БГУИР КП 1-40 02 02 042 ПЗ

Студент: гр. 850702 Турко В. Д.

Руководитель: 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.....	9
2.1 ИСПОЛЬЗУЕМЫЕ ТИПЫ.....	9
2.2 VHDL-ОПИСАНИЕ ПРОЦЕССОРНЫХ ЭЛЕМЕНТОВ	9
2.3 VHDL-ОПИСАНИЕ ПРОЦЕССОРА УМНОЖЕНИЯ.....	10
3 Синтез и моделирование	13
3.1 Синтез принципиальной схемы.....	13
3.2 МОДЕЛИРОВАНИЕ	14
Заключение	16
Список использованных источников	17
ПРИЛОЖЕНИЕ А VHDL-описание типа матрицы и констант	18
ПРИЛОЖЕНИЕ Б VHDL-описание процессорного элемента	19
ПРИЛОЖЕНИЕ В VHDL-описание процессора умножения матриц	20
ПРИЛОЖЕНИЕ Г Код тестовой программы	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 проектирование посвящено умножению матриц, а его темой является «Систолический процессор умножения матриц».

Целью данной курсовой работы является разработка систолического процессора умножения матриц. В соответствии с поставленной целью необходимо разработать VHDL-описание систолического процессора умножения матриц 4 на 4. Такой размер обрабатываемых матриц выбран для удобства отображения графического материала. Однако, как будет показано в разделе 2, полученное VHDL-описание позволяет создавать систолический массив для обработки квадратных матриц любого размера.

1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1 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.

1.2 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 устройства.

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

Умножение матриц $C = AB$ означает вычисление элементов матрицы C в виде:

$$c_{ij} = \sum_{k=1}^n a_{ik}b_{kj}$$

Из этого выражения видно, что вычисления для каждого элемента матрицы C могут быть выполнены одновременно, так как между ними нет никаких зависимостей. Граф зависимостей для алгоритма умножения матриц 4 на 4 представлен на рисунке 1.1, а функциональное описание узла ГЗ – на рисунке 1.2.

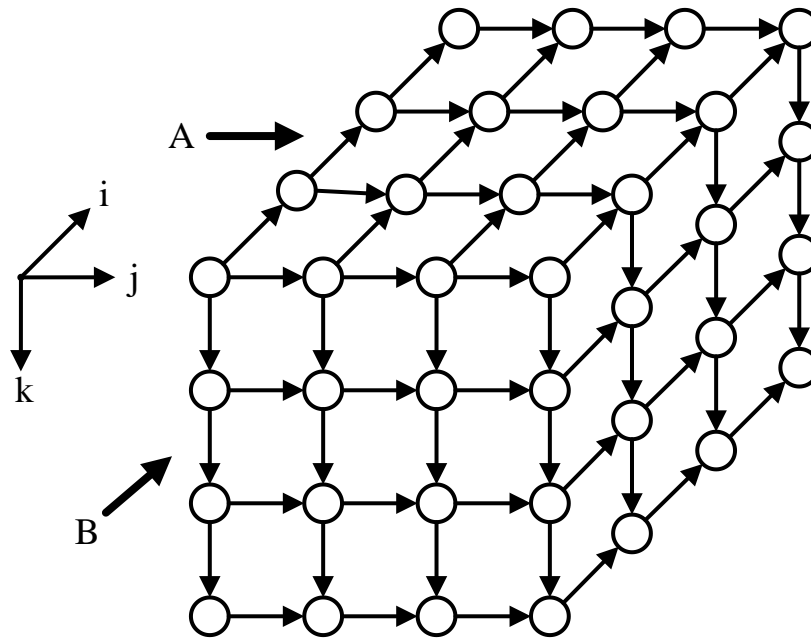


Рисунок 1.1 — ГЗ перемножения матриц

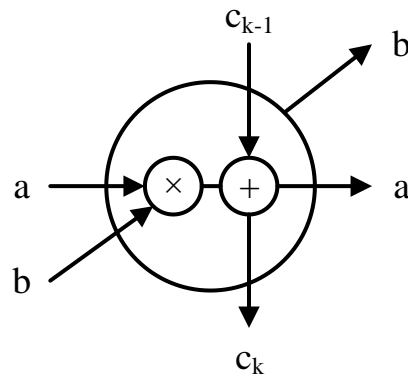


Рисунок 1.2 — Функционирование узла ГЗ для алгоритма умножения матриц

После проекции ГЗ в направлении $[0\ 0\ 1]$ получим ГПС, представленный на рисунке 1.3. Этот ГПС является пространственно-локализованным, но не временно-локализованным; т.е. данные распространяются без задержек, поэтому необходимо произвести ресинхронизацию.

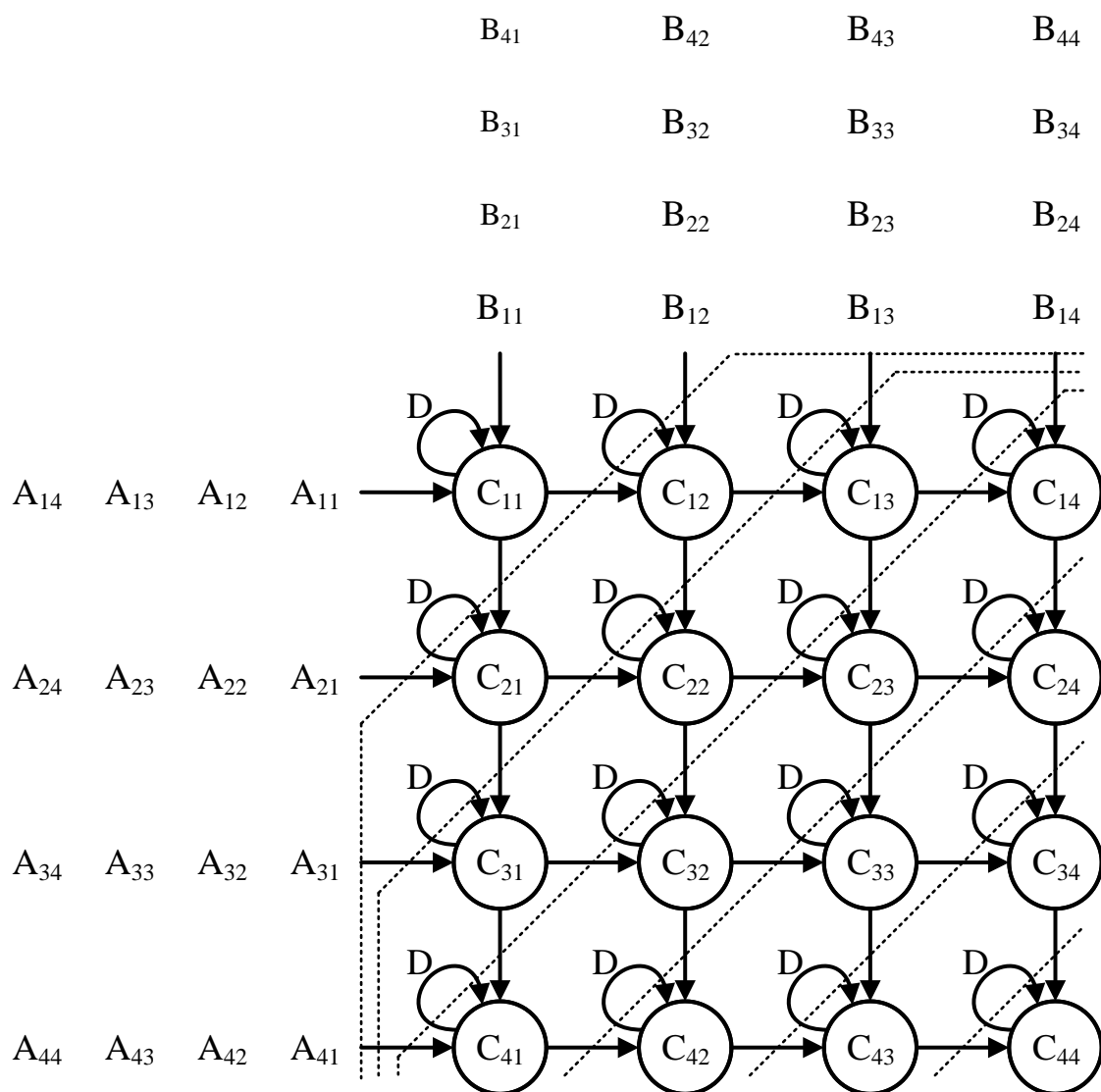


Рисунок 1.3 — ГПС для перемножения матриц 4 на 4

Для дальнейшей разработки систолического массива необходимо преобразование ГПС в систолический массив, иначе говоря, провести систолизацию. Данная процедура включает в себя следующие этапы.

- выбор основных функциональных модулей
- применение правил ресинхронизации
- объединение задержки с функциональным модулем.

В качестве функционального модуля выберем «скалярное умножение», поведение которого описано на рисунке 1.4а. Полученный после объединения задержки с функциональным модулем процессорный элемент показан на рисунке 1.4б.

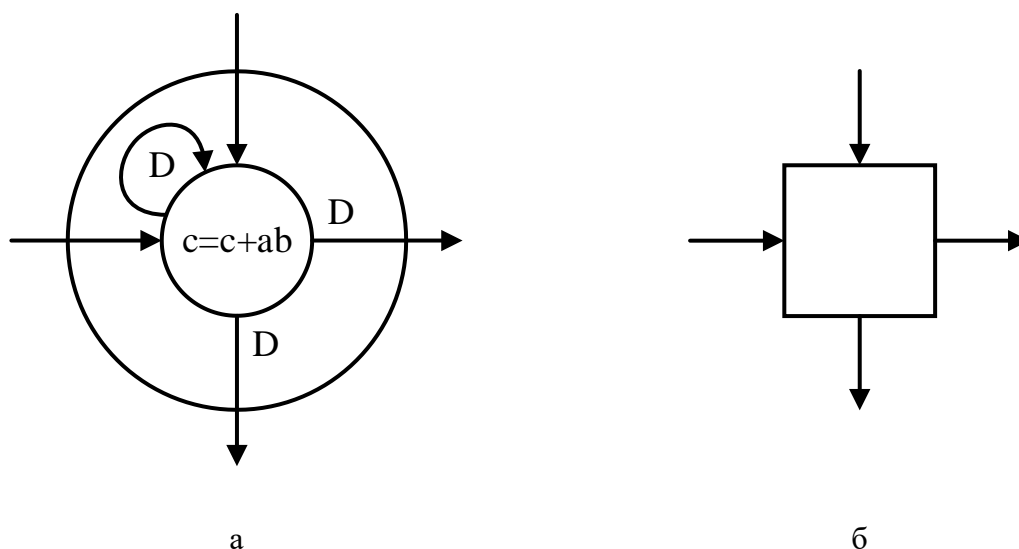


Рисунок 1.4 — Объединение модульной операции с задержкой. а – модульная операция с задержками; б – соответствующий процессорный элемент

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 1.2.

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 1.5.

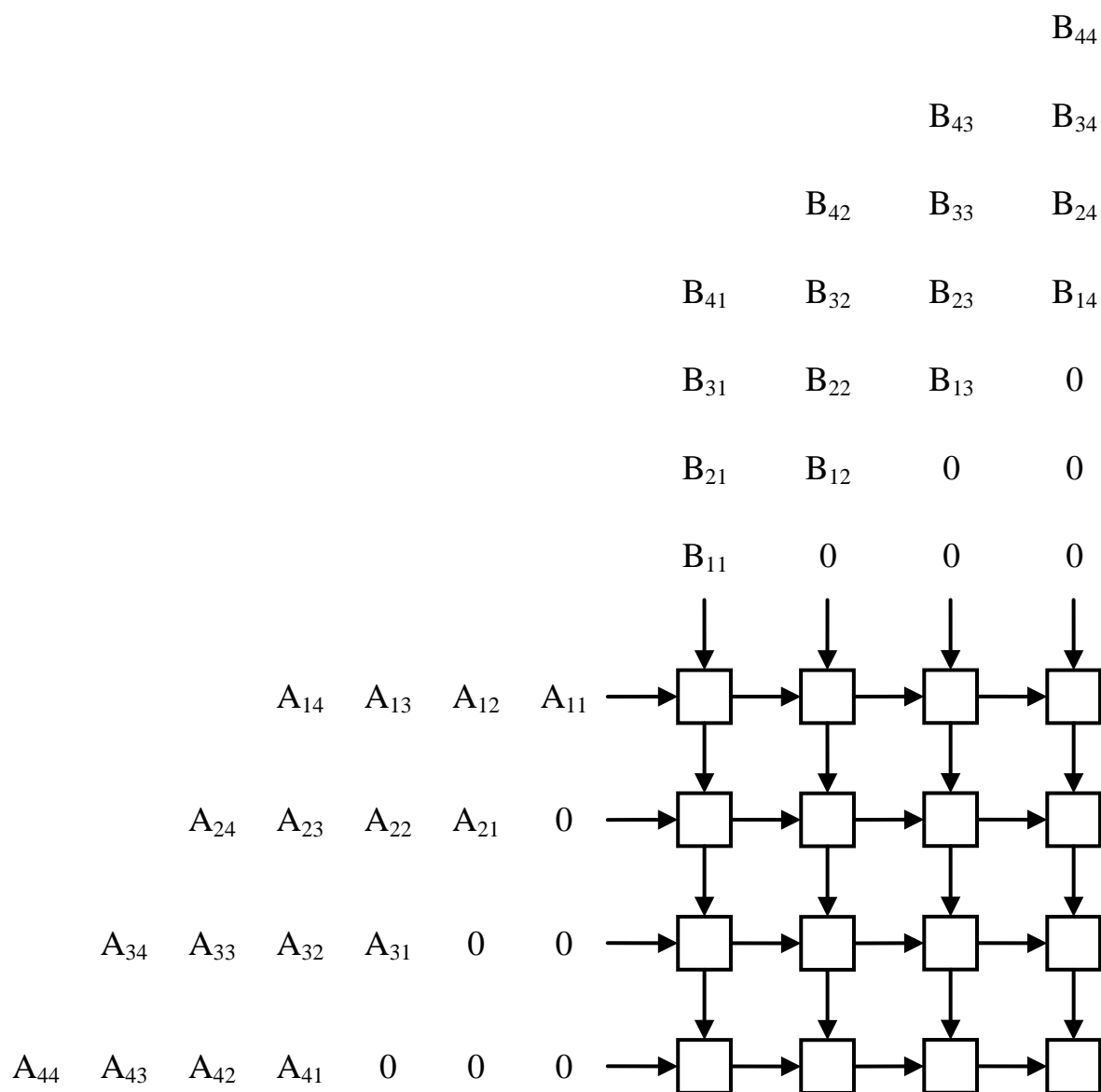


Рисунок 1.5 — Систолический массив для умножения матриц

2 РАЗРАБОТКА VHDL-ОПИСАНИЯ

2.1 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 DOWNT0 0);
  TYPE result_matrix IS ARRAY (POSITIVE RANGE <>, POSITIVE RANGE <>)
    OF STD_LOGIC_VECTOR(RESULT_SIZE DOWNT0 0);
END PACKAGE matrix_package;
```

EL_SIZE и *RESULT_SIZE* – константы, задающие количество зарядов данных, значения которых в данной работе – 4 и 8 соответственно.

2.2 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 *C*, 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 DOWNT0 0) := (OTHERS => '0');
    b : IN STD_LOGIC_VECTOR(EL_SIZE DOWNT0 0) := (OTHERS => '0');
    a_out : OUT STD_LOGIC_VECTOR(EL_SIZE DOWNT0 0) := (OTHERS => '0');
    b_out : OUT STD_LOGIC_VECTOR(EL_SIZE DOWNT0 0) := (OTHERS => '0');
    c : INOUT STD_LOGIC_VECTOR(RESULT_SIZE DOWNT0 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.

2.3 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 *c* 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.

$$\begin{array}{cccc}
 a_{11} & a_{12} & a_{13} & a_{14} \\
 a_{21} & a_{22} & a_{23} & a_{24} \\
 a_{31} & a_{32} & a_{33} & a_{34} \\
 a_{41} & a_{42} & a_{43} & a_{44}
 \end{array}
 \times
 \begin{array}{cccc}
 b_{11} & b_{12} & b_{13} & b_{14} \\
 b_{21} & b_{22} & b_{23} & b_{24} \\
 b_{31} & b_{32} & b_{33} & b_{34} \\
 b_{41} & b_{42} & b_{43} & b_{44}
 \end{array}
 =
 \begin{array}{cccc}
 c_{11} & c_{12} & c_{13} & c_{14} \\
 c_{21} & c_{22} & c_{23} & c_{24} \\
 c_{31} & c_{32} & c_{33} & c_{34} \\
 c_{41} & c_{42} & c_{43} & c_{44}
 \end{array}$$

Рисунок 2.1 — Представление матриц

Перед подачей непосредственно в систолический массив, матрицы A и B 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;
END GENERATE GEN_PROC_ROWS;
```

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

3 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.

3.1 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%.


Device Utilization Summary (estimated values)				
Logic Utilization	Used	Available	Utilization	
Number of Slices	229	20480	1%	
Number of Slice Flip Flops	230	40960	0%	
Number of 4 input LUTs	411	40960	1%	
Number of bonded IOBs	259	333	77%	
Number of MULT18X18s	16	40	40%	
Number of GCLKs	1	8	12%	

Рисунок 3.1 — Результат синтеза умножителя матриц 4-х разрядных чисел 4 на 4.


Device Utilization Summary (estimated values)				
Logic Utilization	Used	Available	Utilization	
Number of Slices	55	20480	0%	
Number of Slice Flip Flops	55	40960	0%	
Number of 4 input LUTs	103	40960	0%	
Number of bonded IOBs	67	333	20%	
Number of MULT18X18s	4	40	10%	
Number of GCLKs	1	8	12%	

Рисунок 3.2 — Результат синтеза умножителя матриц 4-х разрядных чисел 2 на 2.

Device Utilization Summary (estimated values)				[...]
Logic Utilization	Used	Available	Utilization	
Number of Slices	183	20480		0%
Number of Slice Flip Flops	128	40960		0%
Number of 4 input LUTs	338	40960		0%
Number of bonded IOBs	131	333		39%
Number of GCLKs	1	8		12%

Рисунок 3.3 — Результат синтеза умножителя матриц 2-х разрядных чисел 4 на 4.

Device Utilization Summary (estimated values)				[...]
Logic Utilization	Used	Available	Utilization	
Number of Slices	406	20480		1%
Number of Slice Flip Flops	466	40960		1%
Number of 4 input LUTs	746	40960		1%
Number of bonded IOBs	515	333		154%
Number of MULT18X18s	16	40		40%
Number of GCLKs	1	8		12%

Рисунок 3.4 — Результат синтеза умножителя матриц 8-х разрядных чисел 4 на 4.

3.2 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 подробно 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.

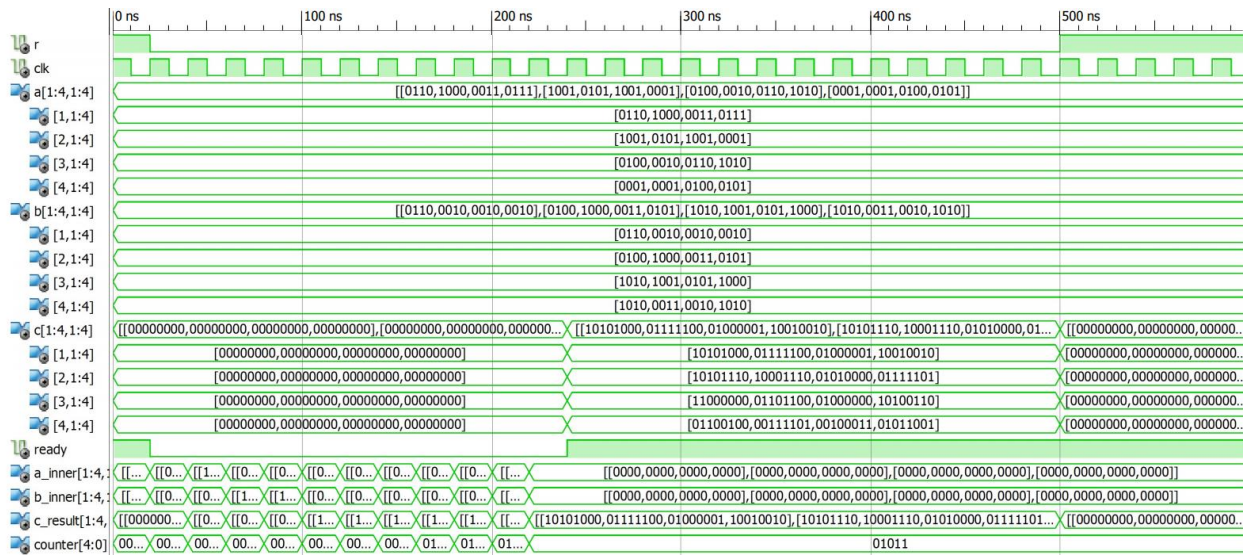


Рисунок 3.5 — Временная диаграмма работы систолического процессора умножения матриц 4 на 4.

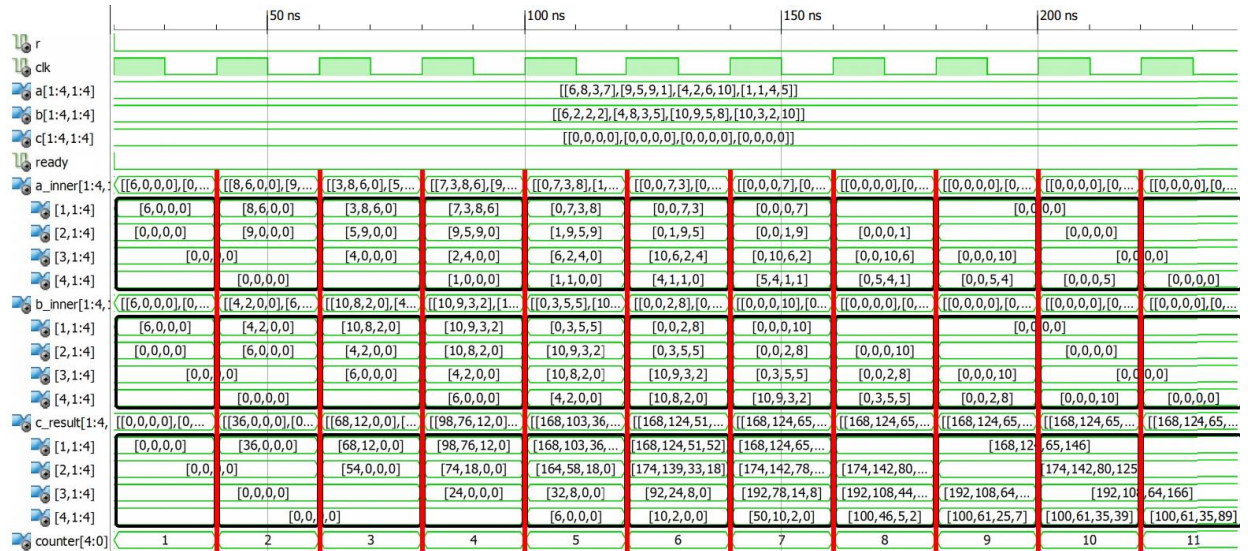


Рисунок 3.6 — Кадры на временной диаграмме для систолического массива умножения матриц 4 на 4.

Как видно из временной диаграммы 3.6, необходимое количество тактов длины T для умножения квадратных матриц размера N на N равно $3N - 1$, а общее время выполнения вычислений – $(3N - 1) * T$.

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 N на N , содержащих M -разрядные элементы, необходимо N^2 процессорных элементов, а также $2N^2$ M -разрядных портов ввода и N^2 $2M$ -разрядных портов вывода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] 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.

ПРИЛОЖЕНИЕ А

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 DOWNT0 0);
    TYPE result_matrix IS ARRAY (POSITIVE RANGE <>, POSITIVE RANGE <>)
        OF STD_LOGIC_VECTOR(RESULT_SIZE DOWNT0 0);
END PACKAGE matrix_package;
```

ПРИЛОЖЕНИЕ Б

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 DOWNT0 0) := (OTHERS => '0');
            b : IN STD_LOGIC_VECTOR(EL_SIZE DOWNT0 0) := (OTHERS => '0');
            a_out : OUT STD_LOGIC_VECTOR(EL_SIZE DOWNT0 0) := (OTHERS => '0');
            b_out : OUT STD_LOGIC_VECTOR(EL_SIZE DOWNT0 0) := (OTHERS => '0');
            c : INOUT STD_LOGIC_VECTOR(RESULT_SIZE DOWNT0 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 DOWNT0 0) := (OTHERS => '0');
        b : IN STD_LOGIC_VECTOR(EL_SIZE DOWNT0 0) := (OTHERS => '0');
        a_out : OUT STD_LOGIC_VECTOR(EL_SIZE DOWNT0 0) := (OTHERS => '0');
        b_out : OUT STD_LOGIC_VECTOR(EL_SIZE DOWNT0 0) := (OTHERS => '0');
        c : INOUT STD_LOGIC_VECTOR(RESULT_SIZE DOWNT0 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;
```

ПРИЛОЖЕНИЕ В

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;

```

ПРИЛОЖЕНИЕ Г

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;
```