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 языке описания аппаратуры

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

VHDL МОДЕЛЬ УСТРОЙСТВА УПРАВЛЕНИЯ ПОВЕДЕНИЕМ «УМНОГО МУРАВЬЯ»

БГУИР КСиС 1-40 02 02 017 ПЗ

Студент: гр. 550701 Шимко М.Д.
Руководитель: Бибило П.Н.

Минск 2017

Содержание

ВВЕДЕНИЕ 3

1 Разработка алгоритма функционирования устройства 4

1.1 Постановка задачи 4

1.2 Выделение подзадач 4

1.3 Алгоритм работы разработанного устройства 5

2 Разработка аналога на языке программирования высокого уровня 8

2.1 Разработка кода программы 8

2.2 Результаты работы прототипа 12

3 Разработка VHDL программ 14

3.1 Разработка кода основной программы 14

3.2 Определение типов и функций в пакете 17

3.3 Разработка кода тестирующей программы 21

4 Моделирование и отладка VHDL программ 24

4.1 Моделирование проекта 24

4.2 Покрытие кода программы 25

5 Разработка синтезируемого варианта VHDL программы 26

5.1 Краткие теоретические сведения 26

5.2 Измененные конструкции в основной программе 26

6 Синтез устройства на элементной базе ПЛИС 29

6.1 Подготовка к синтезу 29

6.2 Экспериментальное исследование зависимости увеличения аппаратной сложности от увеличения числа аргументов функции 30

ЗАКЛЮЧЕНИЕ 32

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33

ВВЕДЕНИЕ

Языки описания аппаратуры и VHDL в частности являются весьма впечатляющими и эффективными средствами проектирования цифровой аппаратуры по сравнению с традиционными методами. Определённое сходство VHDL с языками программирования устанавливает невысокий «порог входа» для новых специалистов. Возможность использовать операторы высокоуровневых языков раздвигают границы классической модели проектирования аппаратуры. В то же время, расценивание языков описания аппаратуры как языков программирования, в корне ошибочно. Специфика и необходимость синтезировать в дальнейшем полученное описание накладывают достаточно строгие ограничения по сравнению с традиционными алгоритмами программирования.

Возможность описывать работу устройства с помощью описания программным кодом даёт выигрыш в скорости разработки устройств самой различной сложности и специфики, существенно упрощается верификация, открывается возможность автоматического тестирования огромных проектов, исключается человеческий фактор. Сходство с языками программирования позволяет «взглянуть на задачу под другим углом». Все эти факторы позитивно сказываются на скорости и качестве разработки.

Данный курсовой проект позволит закрепить знания по разработке на языке VHDL и выполнить практические задачи проектирования. Это положительным образом скажется на навыках проектирования устройств и позволит овладеть методиками комплексной верификации.

1 Разработка алгоритма функционирования устройства

1.1 Постановка задачи

По ограниченному квадратному полю размером 20x20 клеток передвигается муравей. На поле в случайно выбранных клетках находится еда. Муравей может передвигаться в любую из восьми соседних клеток, брать еду из клетки, на которой стоит. При смене позиции муравей оставляет в покинутой клетке некоторое количество феромона. Если муравей кладет феромон в клетку, в которой уже находится феромон, то их количество суммируется.

Муравей обладает информацией о том, что находится в соседних восьми клетках и в клетке, на которой он стоит. Муравей не может выходить за пределы поля.

После того как муравей взял еду, он должен с ней вернуться в клетку, с которой он начал движение, и выполнить в этой части поля действие – положить еду. После этого он может приступить к поиску новой еды. В любой момент времени муравей может нести не более одной единицы еды. Муравей может за один ход либо взять еду из текущей клетки, либо положить еду в муравейник, либо переместиться в соседнюю клетку поля.

1.2 Выделение подзадач

На этом этапе требуется выбрать разбиение задачи системы на подзадачи. Для каждой подзадачи имеет смысл определить, какие входные данные ей потребуются, и какие выходные воздействия система может генерировать. Можно выделить две подзадачи: найти и взять еду, вернуться в муравейник и положить еду. Первую будем назвать задачей A, вторую – задачей B. Для выполнения каждой из этих задач муравей должен знать о наличии еды в текущей и восьми соседних клетках поля, о количестве феромона в текущей и восьми соседних клетках поля, о проходимости восьми соседних клеток поля (находятся ли они за границей поля). Муравей должен иметь возможность

перемещаться в любую из восьми соседних клеток поля, брать еду во время выполнения задачи A и класть еду в муравейник во время выполнения задачи B.

1.3 Алгоритм работы разработанного устройства

Для упрощения дальнейшего описания введем некоторые определения. Так, текущие координаты муравья назовем ant, для подзадачи A (найти и взять еду) goHome будет равно нулю, для подзадачи B (вернуться в муравейник и положить еду) goHome будет равно единице. Еда – состояние сигнала матрицы eat в конкретной ячейке поля (“1” – наличие еды, “0” – отсутствие еды). Феромон – состояние сигнала матрицы pheromone в конкретной ячейке поля.

Непосредственно перед началом выполнения основного алгоритма необходимо провести инициализацию начальных параметров. К таковым относятся матрица еды (eat), матрица феромона (pheromone, крайние строки и столбцы заполняются отрицательными единицами, они и являются указателем на край поля), координаты муравья (next_ant, записываются координаты нижнего правого угла матрицы), сигнал инициализации (initialize, пока не произошла инициализация еды, сигнал равен нулю), случайное число (tmp_random_result).

В условии к выполнению задачи не было сказано, по какому фронту должен работать автомат, был выбран передний фронт. Следовательно, самой первой является проверка на изменение сигнала clk из нуля в единицу, если условие выполняется, то начинается выполнение программы.

Алгоритм работы программы:

1. Проверяется, была ли произведена инициализация матрицы еды. Если нет, то проверяется, есть ли в передаваемой матрице (start_eat)

хотя бы одна клетка с едой. Если да, то происходит инициализация еды во внутреннюю матрицу eat.

2. Если инициализация пройдена успешно, то выполняется загрузка координат восьми соседних клеток в ant_near.
3. В переменную best_way (используется для хранения индекса ant_near) записывается первая клетка, в которой количество феромона не равно отрицательной единице. Обнуляется equal_count (используется для хранения количества соседних клеток с равным числом феромона).
4. Если муравей выполняет подзадачу B, то проверяется, достиг ли он муравейника, если да, то положить еду, сменить задачу B на A. Ожидание переднего фронта. Иначе перейти на шаг 5.
5. Найти клетку среди соседних с максимальным количеством феромона. Результат записать в best_way.
6. Посчитать количество клеток, где число феромона равно best_way. Записать в equal_count.
7. Случайным образом выбрать клетку среди equal_count.
8. Отнять единицу феромона из текущей клетки. Ожидание переднего фронта.
9. Если муравей выполняет подзадачу A, то проверяется, стоит ли он на клетке с едой. Если да, то взять еду, сменить задачу A на B. Ожидание переднего фронта. Иначе перейти на шаг 10.
10. Найти клетку среди соседних, содержащую еду, либо с минимальным положительным количеством феромона. Записать в best_way.
11. Если в best_way содержится еда, то перейти к шагу 14. Иначе посчитать количество клеток с минимальным положительным числом феромона. Записать в equal_count.
12. Случайно выбрать клетку из equal_count.

13. Увеличить число феромона в текущей клетке на две единицы.
14. Обновить позицию муравья (next_ant). Ожидание переднего фронта.

В ходе разработки и тестирования алгоритма было определено, что наилучшей стабильностью обладает алгоритм, при котором число феромона в клетке при поисках еды увеличивается на два, а при возвращении в муравейник уменьшается на единицу. Выбор случайной клетки из соседних, если их приоритеты равны. В таком алгоритме не наблюдается зацикливаний, обеспечивается покрытие (пребывание во всех ячейках поля) муравьем всех клеток на поле. Так же, согласно теории вероятностей, даже, если муравей будет находиться на поле, в котором изначально нет дорожек, он все равно найдет путь в муравейник.

2 Разработка аналога на языке программирования высокого уровня

2.1 Разработка кода программы

Для разработки прототипа был выбран высокоуровневый язык программирования C#. Был организован вывод значений в графическом виде, пошаговое вычисление следующего шага, а так же считывание значений из файла и представление их в графическом виде (используется для верификации VHDL–программы). Стартовое окно программы представлено на рисунке 2.1.1

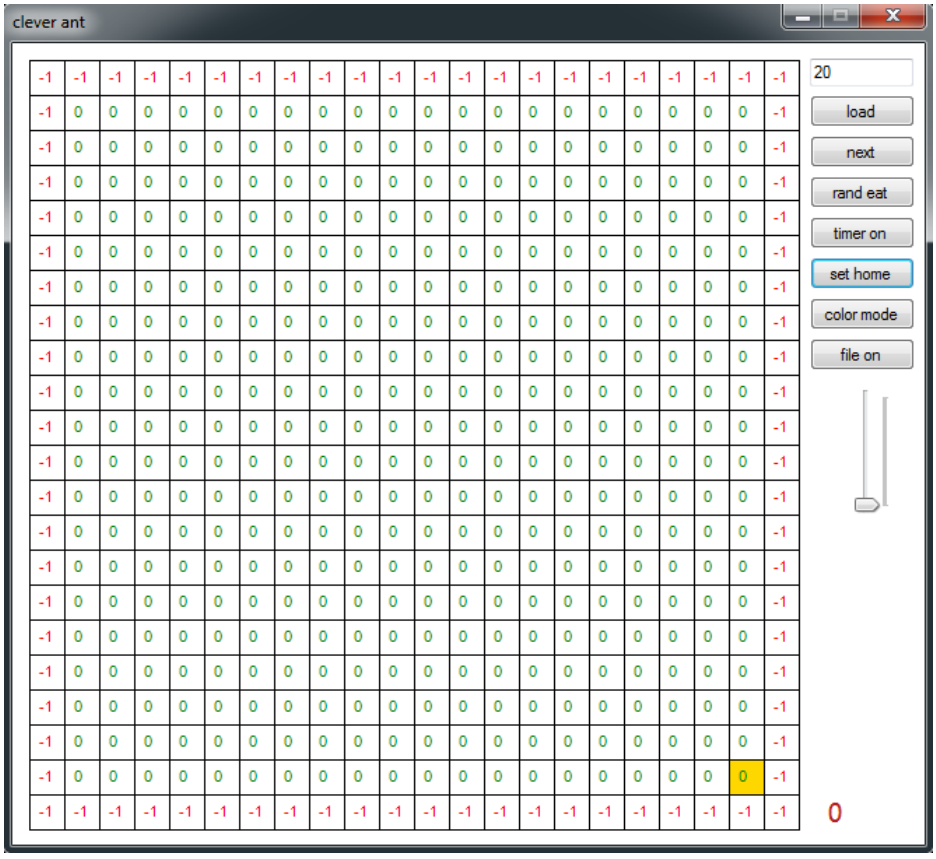

Рисунок 2.1.1 — Стартовое окно программы

Далее будут приведены основные функции программы без кода программирования элементов ввода (поля для ввода, кнопки, слайдеры).

Код функции для расчета следующего шага:

```
private int generateNextStep() {
```

```
    var same_cells = new List<int>();
    int fer_value = 0;
    for (int i = 0; i < 8; i++)
        if (pheromone[ant_near[i, 0], ant_near[i, 1]] != -1) {
            fer_value = i;
            break;
        }
    if (goHome) {
        for (int i = 0; i < 8; i++) {
            if (pheromone[ant_near[i, 0], ant_near[i, 1]] >
pheromone[ant_near[fer_value, 0], ant_near[fer_value, 1]])
                fer_value = i;
        }
    }
    else {
        for (int i = 0; i < 8; i++) {
            if(pheromone[ant_near[i, 0], ant_near[i, 1]] > -1)
                if (eat[ant_near[i, 0] - 1, ant_near[i, 1] - 1])
                    return i;

            if (pheromone[ant_near[i, 0], ant_near[i, 1]] <
pheromone[ant_near[fer_value, 0], ant_near[fer_value, 1]] &&
pheromone[ant_near[i, 0], ant_near[i, 1]] != -1)
                fer_value = i;
        }
    }
    for (int i = 0; i < 8; i++) {
        if (pheromone[ant_near[i, 0], ant_near[i, 1]] ==
pheromone[ant_near[fer_value, 0], ant_near[fer_value, 1]])
            same_cells.Add(i);
    }
    if (same_cells.Count <= 1)
        return fer_value;
    else
        return same_cells[new Random().Next(same_cells.Count)];
}
```

Код функции для обновления позиции муравья:

```
private void updateAnt(int i, int j) {
    if (i <= 0 || j <= 0 || i >= MATRIX_SIZE + 2 - 1 || j >=
MATRIX_SIZE + 2 - 1)
        return;
    if(fileOn) {
        main_pictureBox.Invoke(new Action(() => {
            using (Graphics g =
Graphics.FromHwnd(main_pictureBox.Handle)) {
                g.FillRectangle(Brushes.White, one_cell * ant[1] + 1,
one_cell * ant[0] + 1, one_cell - 1, one_cell - 1);
```

```
                g.FillRectangle(goHome ? Brushes.RoyalBlue :
Brushes.Gold, one_cell * j + 1, one_cell * i + 1, one_cell - 1,
one_cell - 1);
            }
        }));
        setPheromone(ant[0], ant[1], pheromone[ant[0], ant[1]]);
        setPheromone(i, j, pheromone[i, j]);
        ant[0] = i; ant[1] = j;
        return;
    }
    updateEat(ant[0], ant[1], eat[ant[0] - 1, ant[1] - 1]);
    main_pictureBox.Invoke(new Action(() => {
        using (Graphics g =
Graphics.FromHwnd(main_pictureBox.Handle)) {
            g.FillRectangle(goHome ? Brushes.RoyalBlue :
Brushes.Gold, one_cell * j + 1, one_cell * i + 1, one_cell - 1,
one_cell - 1);
        }
    }));
    setPheromone(i, j, pheromone[i, j]);
    ant[0] = i; ant[1] = j;
    ant_near[0, 0] = ant[0] + 1; ant_near[0, 1] = ant[1] + 1;
    ant_near[1, 0] = ant[0] + 1; ant_near[1, 1] = ant[1];
    ant_near[2, 0] = ant[0];     ant_near[2, 1] = ant[1] + 1;
    ant_near[3, 0] = ant[0] + 1; ant_near[3, 1] = ant[1] - 1;
    ant_near[4, 0] = ant[0] - 1; ant_near[4, 1] = ant[1] + 1;
    ant_near[5, 0] = ant[0];     ant_near[5, 1] = ant[1] - 1;
    ant_near[6, 0] = ant[0] - 1; ant_near[6, 1] = ant[1];
    ant_near[7, 0] = ant[0] - 1; ant_near[7, 1] = ant[1] - 1;
}
```

Код функции обновления еды:

```
private void updateEat(int i, int j, bool val) {
    if (i <= 0 || j <= 0 || i > MATRIX_SIZE || j > MATRIX_SIZE ||
!loaded)
        return;
    eat[i - 1, j - 1] = val;
    main_pictureBox.Invoke(new Action(() => {
        using (Graphics g =
Graphics.FromHwnd(main_pictureBox.Handle)) {
            if (val) {
                g.FillRectangle(Brushes.Crimson, one_cell * j + 1,
one_cell * i + 1, one_cell - 1, one_cell - 1);
            }
            else {
                g.FillRectangle(Brushes.White, one_cell * j + 1,
one_cell * i + 1, one_cell - 1, one_cell - 1);
            }
        }
    }));
```

```
    setPheromone(i, j, pheromone[i, j]);
}
```

Код функции изменения значения феромона:

```
private void setPheromone(int i, int j, int val = -1) {
    if (i <= 0 || j <= 0 || i >= MATRIX_SIZE + 2 - 1 || j >=
MATRIX_SIZE + 2 - 1)
        return;
    main_pictureBox.Invoke(new Action(() => {
        using (Graphics g =
Graphics.FromHwnd(main_pictureBox.Handle)) {
            g.DrawString(val.ToString(), new Font("Arial", one_cell /
3), Brushes.Green, one_cell * j + one_cell / 5, one_cell * i +
one_cell / 5);
        }
    }));
}
```

Код функции инициализации:

```
private void button_load_Click(object sender = null, EventArgs e =
null){
if (!fileOn) {
try {
MATRIX_SIZE = Convert.ToInt32(textBox_matrixSize.Text);
}
catch {
loaded = false;
ErrorMsg("Неверная размерность матрицы");
return;
}
}
if (MATRIX_SIZE > 99 || MATRIX_SIZE < 5) {
loaded = false;
ErrorMsg("Неверная размерность матрицы");
return;
}
if(timerOn)//Выключаем таймер, если включен
button_timer_Click();
one_cell = main_pictureBox.Width / (MATRIX_SIZE + 2);
main_pictureBox.Invoke(new Action(() => {
using (Graphics g = Graphics.FromHwnd(main_pictureBox.Handle)) {
g.Clear(Color.White);
}
}));
if (!fileOn) {
pheromone = new int[MATRIX_SIZE + 2, MATRIX_SIZE + 2];
eat = new bool[MATRIX_SIZE, MATRIX_SIZE];

for (int i = 0; i < MATRIX_SIZE + 2; i++)
for (int j = 0; j < MATRIX_SIZE + 2; j++) {
```

```
if (i == 0 || i == MATRIX_SIZE + 2 - 1 || j == 0 || j == MATRIX_SIZE
+ 2 - 1)
pheromone[i, j] = -1;
else
pheromone[i, j] = 0;
}
}
for (int i = 0; i < MATRIX_SIZE + 2; i++) {
for (int j = 0; j < MATRIX_SIZE + 2; j++) {
main_pictureBox.Invoke(new Action(() => {
using (Graphics g = Graphics.FromHwnd(main_pictureBox.Handle)) {
using (var frame_line = new Pen(Color.Black, 1)) {
g.DrawRectangle(frame_line, one_cell * i, one_cell * j, one_cell,
one_cell);
if (fileOn && i > 0 && j > 0 && i <= MATRIX_SIZE && j <= MATRIX_SIZE)
{
loaded = true;
updateEat(i, j, eat[i - 1, j - 1]);
}
if (i == 0 || i == MATRIX_SIZE + 2 - 1 || j == 0 || j == MATRIX_SIZE
+ 2 - 1)
g.DrawString("-1", new Font("Arial", one_cell / 3), Brushes.Red,
one_cell * j + one_cell / 5, one_cell * i + one_cell / 5);
}
}
}));
setPheromone(i, j, pheromone[i, j]);
}
}
if (fileOn) {
ant[0] = startX; ant[1] = startY;
}
else {
ant[0] = 1; ant[1] = 1;
startX = startY = 1;
}
eatCount = 0;
label_eatCount.Text = "0";
goHome = false;
updateAnt(startX, startY);
loaded = true;
}
```

2.2 Результаты работы прототипа

Программа тестировалась на разных наборах еды, а так же при разной размерности поля (от 5x5 до 50x50). Опытным путем было выяснено, что чем больше еды на поле, тем более равномерно распределяется количество

феромона по клеткам. Это можно наблюдать на рисунках: 2.2.1, 2.2.2, 2.2.3, 2.2.4.

При большем количестве еды на поле уменьшается вероятность того, что муравей окажется в зоне, где количество феромона в соседних клетках равно нулю. Для формирования устойчивых дорожек к муравейнику рекомендуемое заполнение всего поля едой не менее 30%.

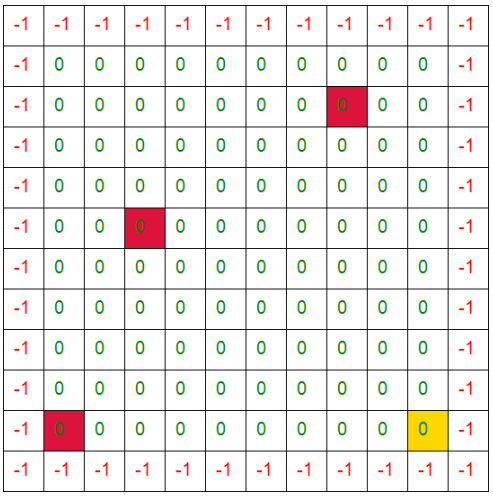

Рисунок 2.2.1 – Заполнение 3%
Входные данные

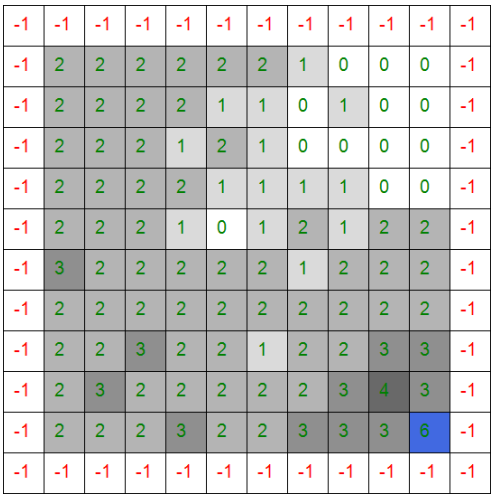

Рисунок 2.2.2 – Заполнение 3%
Выходные данные

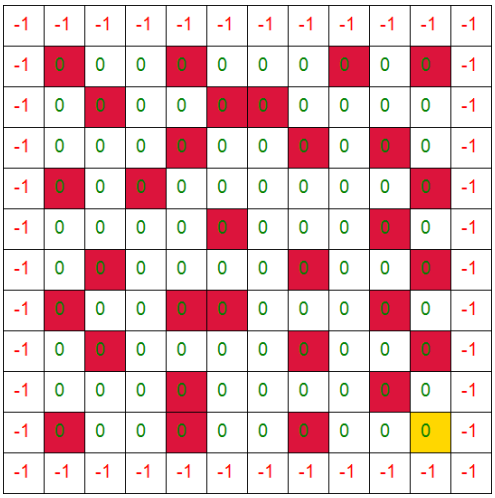

Рисунок 2.2.3 – Заполнение 30%
Входные данные

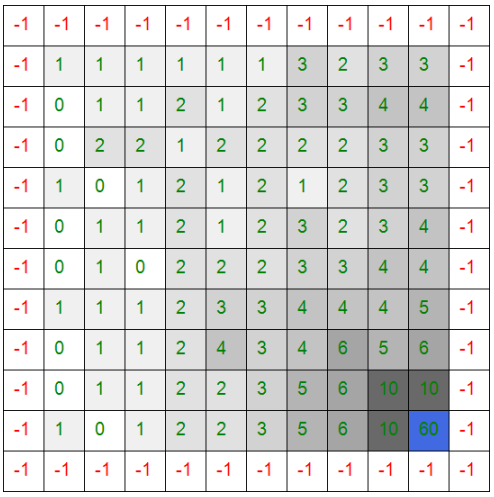

Рисунок 2.2.4 – Заполнение 30%
Выходные данные

Как можно видеть, на рисунке 2.2.4 распределение феромона в клетках более правильное, чем на рисунке 2.2.2. Это связанно с тем, что муравей быстрее находит еду и путь к муравейнику более прямолинейный.

3 Разработка VHDL программ

3.1 Разработка кода основной программы

На вход VHDL модели поступает синхросигнал clk, а так же матрица еды start_eat типа eat_arr (см. 3.2). Выходной сигнал ant – массив из двух элементов, где первый – это номер строки, а второй – это номер столбца на котором находится муравей. Блок-схема алгоритма представлена на рисунке 3.2.1.

Программа состоит из двух взаимодействующих процессов: NS, REG. Процесс NS определяет внутренний сигнал next_ant, который является входным для процесса REG. В списке чувствительности процесса REG имеется только сигнал clk, который соответствует моментам времени срабатывания автомата.

Полный код разрабатываемой программы:

```
Library IEEE, work;
Use IEEE.std_logic_1164.all;
Use work.package1.all;
entity model is
  port(clk : in std_logic;
    start_eat : in eat_arr;
    ant : out ant_arr);
end model;
architecture struct of model is
signal next_ant : ant_arr := (0 => MATRIX_SIZE-2, 1 =>
MATRIX_SIZE-2);
signal eat: eat_arr;
signal pheromone : pheromone_arr := (
  0 => (others => -1),
  MATRIX_SIZE-1 => (others => -1),
  others => (0 => -1, MATRIX_SIZE-1 => -1, others => 0)
);
signal goHome : std_logic := '0';
signal initialize : std_logic := '0';
signal tmp_random_result: integer := 1;
signal t_next_ant: std_logic := '0';
begin
  NS: process(t_next_ant, start_eat)
    constant startPos : ant_arr := (0 => MATRIX_SIZE-2, 1 =>
MATRIX_SIZE-2);
    variable ant_near : ant_near_arr;
    variable best_way : natural range 0 to 7;
    variable equal_count : integer range 0 to 8 := 0;
    variable int_rand : integer range 0 to 7;
```

```
  begin
    if(initialize = '0') then
      if(isEatExist(start_eat) = '1') then
        eat <= start_eat;
        initialize <= '1';
      end if;
    else
      ant_near := (
        0 => (next_ant(0) + 1, next_ant(1) + 1),
        1 => (next_ant(0) + 1, next_ant(1)),
        2 => (next_ant(0),     next_ant(1) + 1),
        3 => (next_ant(0) + 1, next_ant(1) - 1),
        4 => (next_ant(0) - 1, next_ant(1) + 1),
        5 => (next_ant(0),     next_ant(1) - 1),
        6 => (next_ant(0) - 1, next_ant(1)),
        7 => (next_ant(0) - 1, next_ant(1) - 1)
      );
      for i in 0 to 7 loop
        if(pheromone(ant_near(i)(0),ant_near(i)(1)) >= 0) then
          best_way := i;
          exit;
        end if;
      end loop;
      equal_count := 0;
      if(goHome = '1') then
        if(next_ant(0) = startPos(0) and next_ant(1) =
startPos(1)) then
          goHome <= '0';
        else
          for i in 0 to 7 loop
            if(pheromone(ant_near(i)(0),ant_near(i)(1)) >
pheromone(ant_near(best_way)(0),ant_near(best_way)(1))) then
              best_way := i;
            end if;
          end loop;
          for i in 0 to 7 loop
            if(pheromone(ant_near(i)(0),ant_near(i)(1)) =
pheromone(ant_near(best_way)(0),ant_near(best_way)(1))) then
              equal_count := equal_count + 1;
            end if;
          end loop;
          tmp_random_result <= random(tmp_random_result);
          int_rand := getMod(tmp_random_result, equal_count);
          for i in 0 to 7 loop
            if(pheromone(ant_near(i)(0),ant_near(i)(1)) =
pheromone(ant_near(best_way)(0),ant_near(best_way)(1))) then
              if(int_rand = 0) then
                best_way := i;
                exit;
              else
                int_rand := int_rand - 1;
```

```
            end if;
          end if;
        end loop;
        if(pheromone(next_ant(0), next_ant(1)) > 0) then
          pheromone(next_ant(0), next_ant(1)) <=
pheromone(next_ant(0), next_ant(1)) - 1;
        end if;
        next_ant <= (ant_near(best_way)(0),
ant_near(best_way)(1));
      end if;
    else
      if(eat(next_ant(0)-1)(next_ant(1)-1) = '1') then
        goHome <= '1';
        eat(next_ant(0)-1)(next_ant(1)-1) <= '0';
      else
        for i in 0 to 7 loop
          if((pheromone(ant_near(i)(0),ant_near(i)(1)) <=
pheromone(ant_near(best_way)(0),ant_near(best_way)(1))) and
(pheromone(ant_near(i)(0),ant_near(i)(1)) /= -1)) then
            best_way := i;
            if(eat(ant_near(i)(0)-1)(ant_near(i)(1)-1) = '1')
then
              exit;
            end if;
          end if;
        end loop;
        if(eat(ant_near(best_way)(0)-1)(ant_near(best_way)(1)-1)
= '0') then
          for i in 0 to 7 loop
            if(pheromone(ant_near(i)(0),ant_near(i)(1)) =
pheromone(ant_near(best_way)(0),ant_near(best_way)(1))) then
              equal_count := equal_count + 1;
            end if;
          end loop;
            tmp_random_result <= random(tmp_random_result);
            int_rand := getMod(tmp_random_result,
equal_count);
            for i in 0 to 7 loop
            if(pheromone(ant_near(i)(0),ant_near(i)(1)) =
pheromone(ant_near(best_way)(0),ant_near(best_way)(1))) then
              if(int_rand = 0) then
                best_way := i;
                exit;
              else
                int_rand := int_rand - 1;
              end if;
            end if;
          end loop;
        end if;
        pheromone(next_ant(0), next_ant(1)) <=
pheromone(next_ant(0), next_ant(1)) + 2;
```

```
          next_ant <= (ant_near(best_way)(0),
ant_near(best_way)(1));
        end if;
      end if;
    end if;
  end process;
  REG: process(clk)
  begin
    if(clk'event and clk = '1') then
      ant <= next_ant;
      t_next_ant <= not t_next_ant;
    end if;
  end process;
end struct;
```

3.2 Определение типов и функций в пакете

В данном проекте использован пакет package1, который содержит определения для нескольких типов, а так же функций random, getMod и isEatExist.

Полный код пакета package1:

```
Library IEEE;
Use IEEE.std_logic_1164.all;
package package1 is
  constant dimension : natural := 10;
  constant MATRIX_SIZE: natural := dimension + 2;
  type ant_arr is array (natural range 0 to 1) of integer range 0
to MATRIX_SIZE - 1;
  type ant_near_arr is array (natural range 0 to 7) of ant_arr;
  type pheromone_arr is array (natural range 0 to MATRIX_SIZE - 1,
natural range 0 to MATRIX_SIZE - 1) of integer range -1 to 32767;
  type eat_arr is array (natural range 0 to MATRIX_SIZE - 1 - 2)
of std_logic_vector(0 to MATRIX_SIZE - 1 - 2);
  function random(prev_num: integer) return integer;
  function getMod(num: integer; max_val: integer) return integer;
  function isEatExist(eat: eat_arr) return std_logic;
end package1;
package body package1 is
  function random(prev_num: integer) return integer is
  begin
    return ((5**13)*(prev_num)) mod 1073741824;
  end random;
  function getMod(num: integer; max_val: integer) return integer
is
  variable t_num: integer range -8 to 7 := 0;
```

```
  variable result: integer range 0 to 7 := 0;
  begin
    t_num := num mod 8;
    for i in 0 to 7 loop
      if(t_num < 0) then
        exit;
      end if;
      result := t_num;
      t_num := t_num - max_val;
    end loop;
    return result;
  end getMod;
  function isEatExist(eat: eat_arr) return std_logic is
  variable result: std_logic := '0';
  begin
    for i in 0 to MATRIX_SIZE-1-2 loop
      for j in 0 to MATRIX_SIZE-1-2 loop
        if(eat(i)(j) = '1') then
          result := '1';
        end if;
      end loop;
    end loop;
    return result;
  end isEatExist;
end package body package1;
```

Описание констант и типов определенных в пакете:

1. dimension – задает размерность полезной части матрицы.
2. MATRIX_SIZE – содержит полную размерность матрицы (добавляются границы, феромон в которых равен отрицательной единице).
3. ant_arr – содержит координаты муравья.
4. ant_near_arr – массив из восьми ant_arr, содержит координаты соседних клеток.
5. pheromone_arr – двумерный массив типа integer. Максимальное значение ограничено пятнадцатью битами. Хранит значение феромона в клетке.
6. eat_arr – двумерный массив, содержащий информацию о том, есть ли еда в клетке.

Описание функций определенных в пакете:

1. Функция random. Генерирует псевдослучайное число на основе предыдущего. Используется линейный конгруэнтный метод, в данном методе каждое число последовательности генерируется по формуле $k_i = (a*k_{i-1}+b) \bmod c$. Недостатком данного метода является то, что числа в последовательности обладают периодичностью. Конкретно в нашем случае это не важно, подобрав такие числа, что $a = 5^{13}$, $b = 0$, $c = 2^{30}$, получается достаточно большой период, при котором основной алгоритм работает корректно. Стоит заметить, что чем больше c, тем больше период генерируемых чисел, поэтому наилучшим вариантом было бы взять максимальное положительное число: 2^{31}-1. Но так как синтезатор умеет делить только операнды, кратные степени двойки, было принято решение использовать тридцатую степень.
2. Функция getMod. Так как синтезатор позволяет делить только числа кратные степени двойки, была написана функция для генерирования числа от нуля до семи на основе числа, возвращаемого функцией random. Дело в том, что если все соседние клетки муравья с равным количеством феромона, то согласно алгоритму программы нужно выбрать одну клетку из восьми. Следовательно, используются младшие три бита числа num (максимальное значение, когда все три бита равны единице – семь), после чего с помощью последовательного вычитания делителя находится остаток от деления на максимальное число, которое нужно получить. Функция возвращает число в диапазоне от нуля до max_val-1.
3. Функция isEatExist. Проверяет, инициализирована ли матрица еды (eat). Дело в том, что в тестирующей программе значение сигналу eat присваивается не при определении, а при считывании из файла по синхросигналу. Таким образом, изначально в модель подается пустая матрица, а только после инициализированная. Эта функция позволяет определить, были ли значения получены моделью и можно продолжить работу, либо быть в режиме ожидания инициализации.

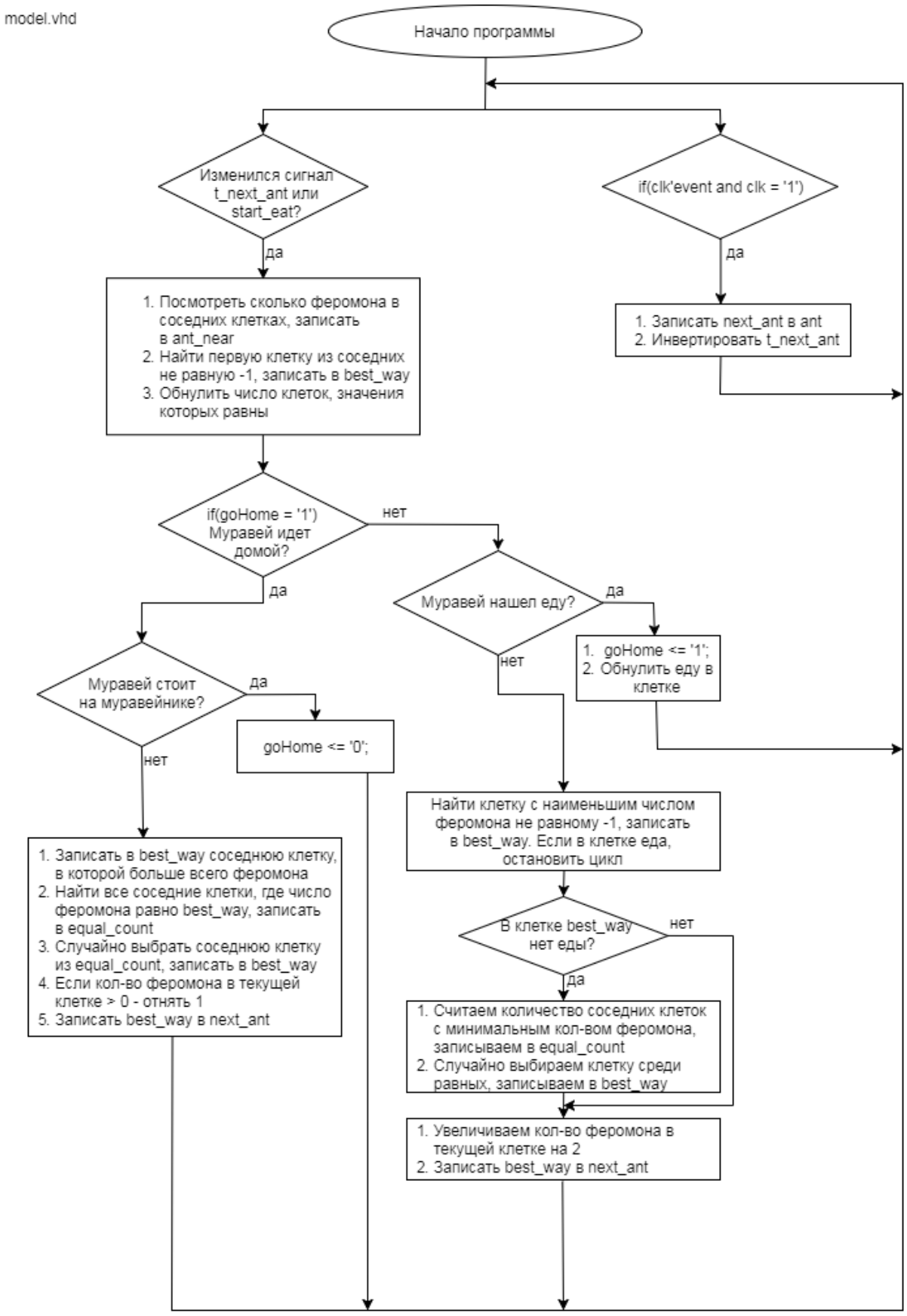

Рисунок 3.2.1 — Блок-схема основной программы

3.3 Разработка кода тестирующей программы

Тестирующая программа используется главным образом для загрузки исходных данных из файла input_data_file.txt и записи результатов в файл output_data_file.txt. Блок-схема представлена на рисунке 3.3.1.

Формат исходного файла: в каждой строке записаны бит-вектора, количество строк и длина бит-вектора в строке равны размерности матрицы.

Формат генерируемого файла: первая строка – строка исходной матрицы еды, где элементы матрицы записываются построчно сверху вниз. Вторая строка – строка матрицы феромона, записывается по принципу выше. Все последующие строки являются координатами муравья, где первое число – строка, а второе число – столбец матрицы.

Алгоритм тестирующей программы:

1. Если еда еще не была загружена из файла, то загрузить.
2. Если передний фронт, то проверить были ли сохранены строки в файл с информацией о матрице с едой, матрице с феромоном. Если нет, то сохранить, иначе сохранить текущие координаты муравья.

Полный код тестирующей программы:

```
Library IEEE, work;
Use IEEE.std_logic_1164.all;
Use work.package1.all;
Use STD.textio.all;
entity test is
end test;
architecture struct of test is
component model is
  port(clk : in std_logic;
    start_eat : in eat_arr;
    ant : out ant_arr);
end component;
signal clk : std_logic;
signal eat : eat_arr;
signal ant : ant_arr;
signal initialize, Eatinitialize: std_logic := '0';
begin
  p1 : model port map (clk, eat, ant);
  log_process: process(clk)
```

```
    variable indata_line : line;
    file input_data_file : text open read_mode is
"input_data_file.txt";
    variable outdata_line : line;
    file output_data_file : text open write_mode is
"output_data_file.txt";
    variable tmp_eat : bit_vector(0 to MATRIX_SIZE-1-2);
  begin
    if(Eatinitialize = '0') then
      for i in 0 to MATRIX_SIZE-1-2 loop
          readline (input_data_file, indata_line);
          read (indata_line, tmp_eat);
          eat(i) <= To_StdLogicVector(tmp_eat);
          Eatinitialize <= '1';
      end loop;
    end if;
    if(clk'event and clk = '1') then
      if(initialize = '0') then
        initialize <= '1';
        for i in 0 to MATRIX_SIZE-1-2 loop
          for j in 0 to MATRIX_SIZE-1-2 loop
            if(eat(i)(j) = '1') then
              write(outdata_line, 1);
            else
              write(outdata_line, 0);
            end if;
            write(outdata_line, string'(" "));
          end loop;
        end loop;
        writeline(output_data_file,outdata_line);
        for i in 0 to MATRIX_SIZE-1 loop
          for j in 0 to MATRIX_SIZE-1 loop
            if(i = 0 or j = 0 or i = MATRIX_SIZE-1 or j =
MATRIX_SIZE-1) then
              write(outdata_line, string'("-1 "));
            else
              write(outdata_line, string'("0 "));
            end if;
          end loop;
        end loop;
        writeline(output_data_file,outdata_line);
      elsif(initialize = '1') then
        write(outdata_line,ant(0));
        write(outdata_line,string'(" "));
        write(outdata_line,ant(1));
        writeline(output_data_file,outdata_line);
      end if;
    end if;
  end process;
end struct;
```

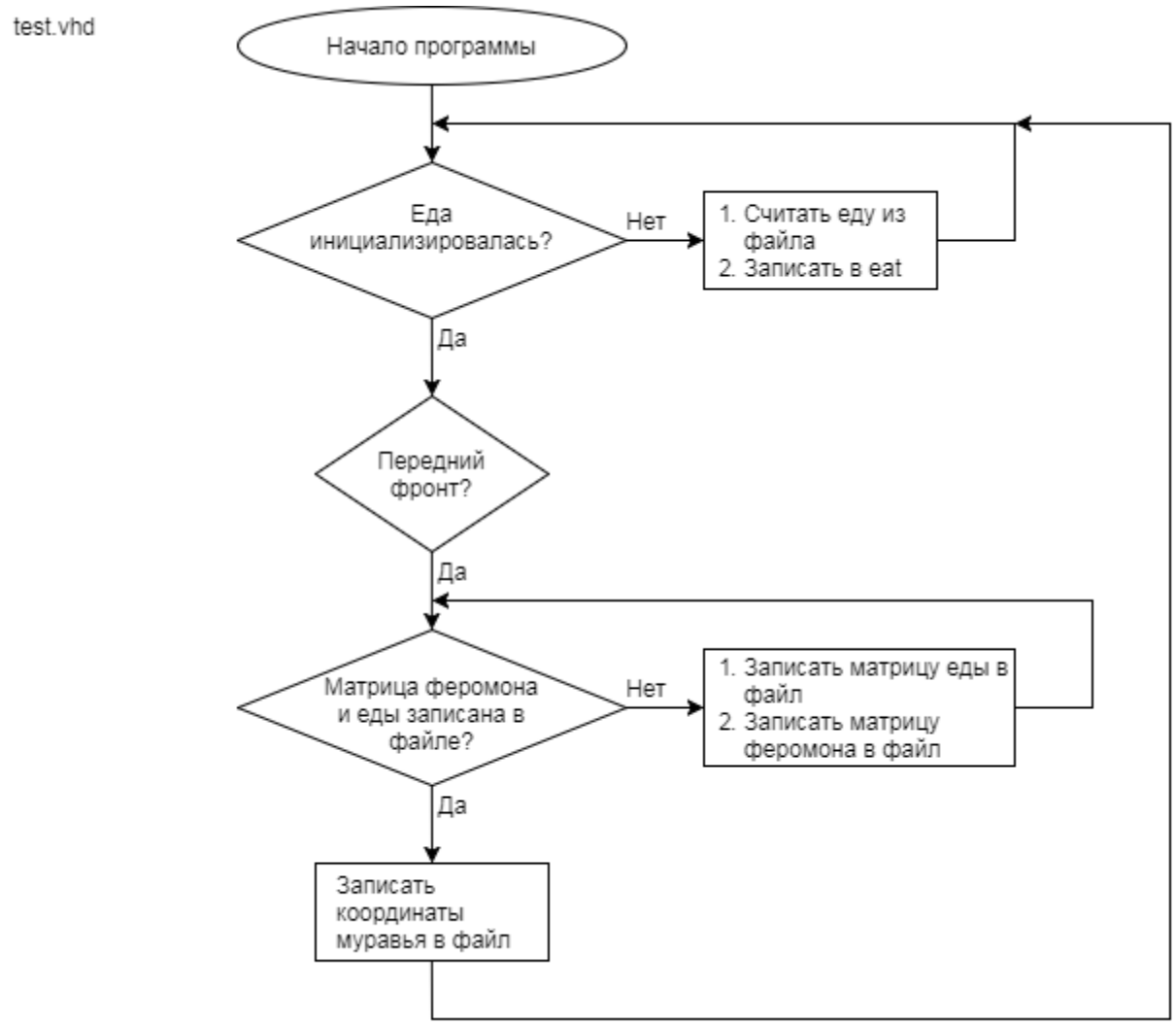

Рисунок 3.3.1 — Блок-схема тестирующей программы

Полный код TCL-скрипта:

```
vsim -coverage -novopt work.test
add wave clk ant
force clk 0 0, 1 25 -repeat 50
run 10000
coverage report -html -htmldir covhtmlreport -threshL 50 -threshH
90
```

4 Моделирование и отладка VHDL программ

4.1 Моделирование проекта

Проведем моделирование устройства, чтобы удостовериться в корректности его работы. В процессе разработки моделирование проводилось неоднократно, на различных наборах и при различных размерностях матрицы.

Для моделирования была использована среда ModelSim, написана тестирующая программа, а так же TCL–скрипт (для генерации синхроимпульса).

В данном конкретном примере представлены результаты моделирования для матрицы размерностью 10x10 и следующим расположением еды:

```
1000101000
0000000000
0010000000
1000101000
1000000000
0000000000
1000101010
0000000000
0101000101
0000011100
```

В данной матрице порядковый номер бита соответствует порядковому номеру клетки, 1 – есть еда в клетке, 0 – нет еды в клетке.

Временная диаграмма представлена на рисунке 4.1.1.

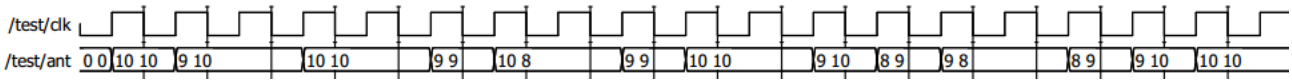

Рисунок 4.1.1 — Временная диаграмма

На временной диаграмме видно, что изначально координаты муравья равны (0,0), а, когда сигнал clk переходит в единицу, координаты становятся равными (10,10), что равно координатам муравейника. Координаты получились равными сразу (10,10), а не (9,10), т.к. в первом такте производилась

инициализация всех сигналов, а не расчет следующей координаты, а уже со второго такта можно видеть, что координата обновилась.

4.2 Покрытие кода программы

Таблицы покрытия кода программы представлены на рисунках 4.2.1 и 4.2.2.

Design Unit Coverage Details:

Weighted Average:			100.0%
Coverage Type	**Bins**	**Hits**	**Coverage (%)**
Statement	14	14	100.0%
Branch	4	4	100.0%

Рисунок 4.2.1 — Покрытие package1

Design Unit Coverage Details:

Weighted Average:			97.8%
Coverage Type	**Bins**	**Hits**	**Coverage (%)**
Statement	46	46	100.0%
Branch	35	35	100.0%
Condition	6	6	100.0%
Toggle	9	8	88.9%

Рисунок 4.2.2 — Покрытие model

Как можно видеть, код model покрыт не полностью, не покрыто одно переключение. Более подробно о том, какой сигнал не был покрыт, показано на рисунке 4.2.3.

Signal / Value	Hits						Status
	0L->1H	1H->0L	0L->Z	Z->1H	1H->Z	Z->0L	
initialize	1	0	--	--	--	--	0.0%

Рисунок 4.2.3 — Непокрытое переключение

Сигнал initialize по логике программы переключается один раз из нуля в единицу, т.к. инициализация всех сигналов происходит только один раз в начале программы, поэтому переключения назад в ноль нет.

5 Разработка синтезируемого варианта VHDL программы

5.1 Краткие теоретические сведения

Производя синтез устройства, необходимо учитывать определённые ограничения, накладываемые технологией синтезирования. Есть определённый список конструкций, которые невозможно синтезировать.

Основные несинтезируемые конструкции:

- Не допускается использование типа real.
- Не допускается использование переменных в цикле loop.
- Операции над файлами не поддерживаются при синтезе.
- Операция деления (так же mod, rem) не поддерживается при синтезе.
- Не допускается использование функций для работы с файлами.

5.2 Измененные конструкции в основной программе

В ходе проектирования системы был допущен ряд ошибок, которые были исправлены, чтобы синтезировать устройство. Стоит заметить, что для моделирования можно использовать гораздо более разные и универсальные конструкции, однако при синтезе они могут не работать.

Первая версия программы работала по алгоритму, указанному на рисунке 5.2.1. Как можно видеть, использовался единственный процесс, в котором проверялся как задний, так и передний фронт синхросигнала. Код процесса NS выполнялся по заднему фронту сигнала clk, а REG по переднему. Результаты при моделировании были те же, однако синтезатор сообщал об ошибке: *"ERROR:Xst:827 - "D:/GitHub/VHDL/course/model.vhd" line 32: Signal eat<0><0> cannot be synthesized, bad synchronous description. The description style you are using to describe a synchronous element (register, memory, etc.) is not supported in the current software release."*. Использовался синтезатор Xilinx.

Чтобы исправить ошибку, было принято решение разделить единственный процесс на два: NS, REG. Создать временный сигнал t_next_ant, который инвертируется при каждом вызове REG и тем самым вызывает процесс NS (внешний сигнал ant нельзя использовать, т.к. он выходной). Список чувствительности процесса NS: t_next_ant, start_eat.

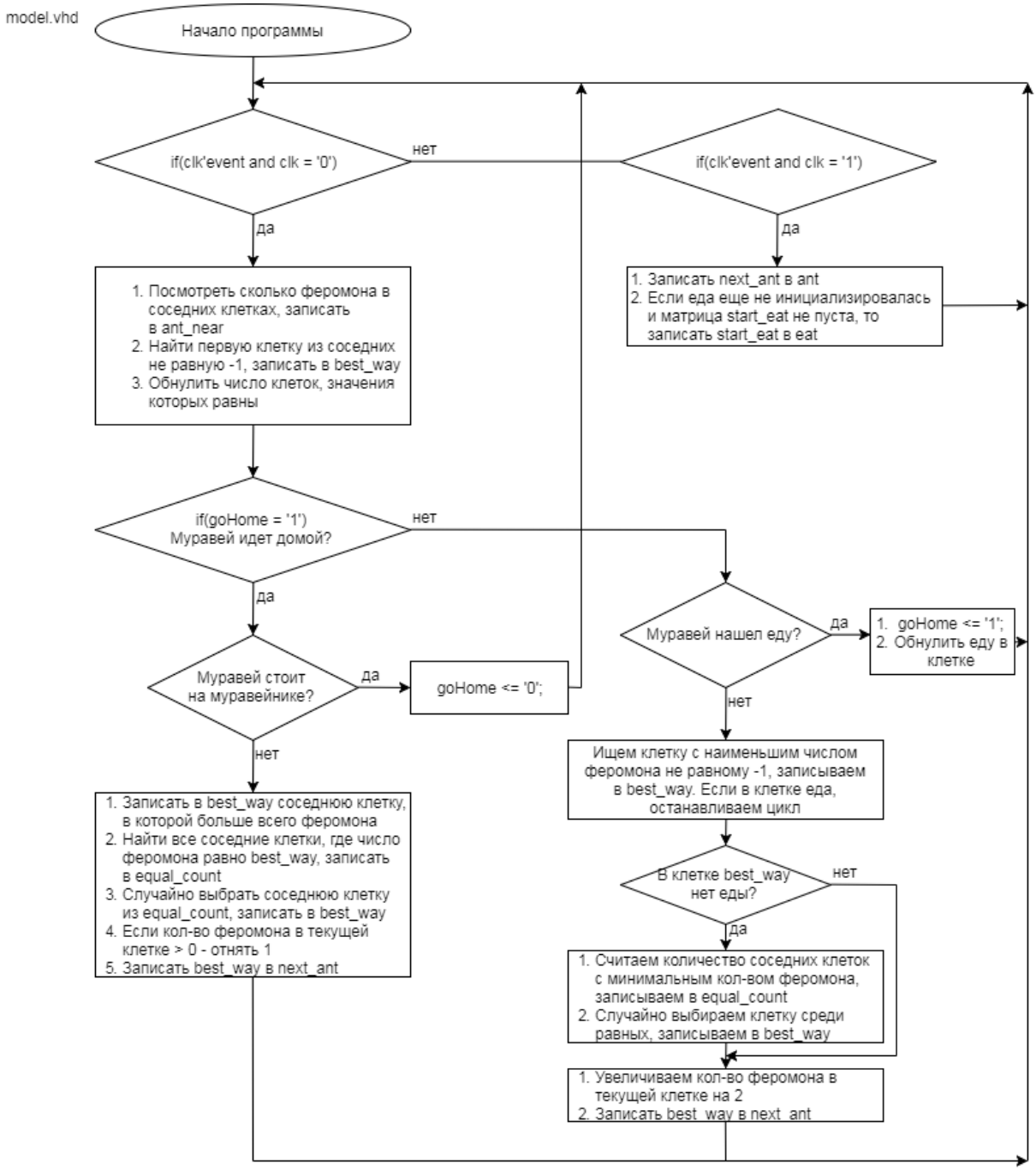

Рисунок 5.2.1 — Блок-схема первой версии программы

В первой версии программы была использована функция uniform для генерации случайного вещественного числа в диапазоне от нуля до единицы. Так как число имеет тип real, была написана своя функция для генерации случайных целых чисел (см. 3.2).

Из основной модели так же были убраны все функции для работы с файлами (некоторые использовались для отладки). Вся работа с файлами перенесена в тестирующую программу.

6 Синтез устройства на элементной базе ПЛИС

6.1 Подготовка к синтезу

Синтез проводился в среде Xilinx ISE при параметрах, представленных на рисунке 6.1.1. Размерность матрицы была установлена в значение 2. В качестве базы синтеза был выбран, согласно условию, Spartan3-1000. Данный выбор обусловлен тем, что конструкции, синтезированные на базе Spartan3, удобно переводить на отечественную базу синтеза.

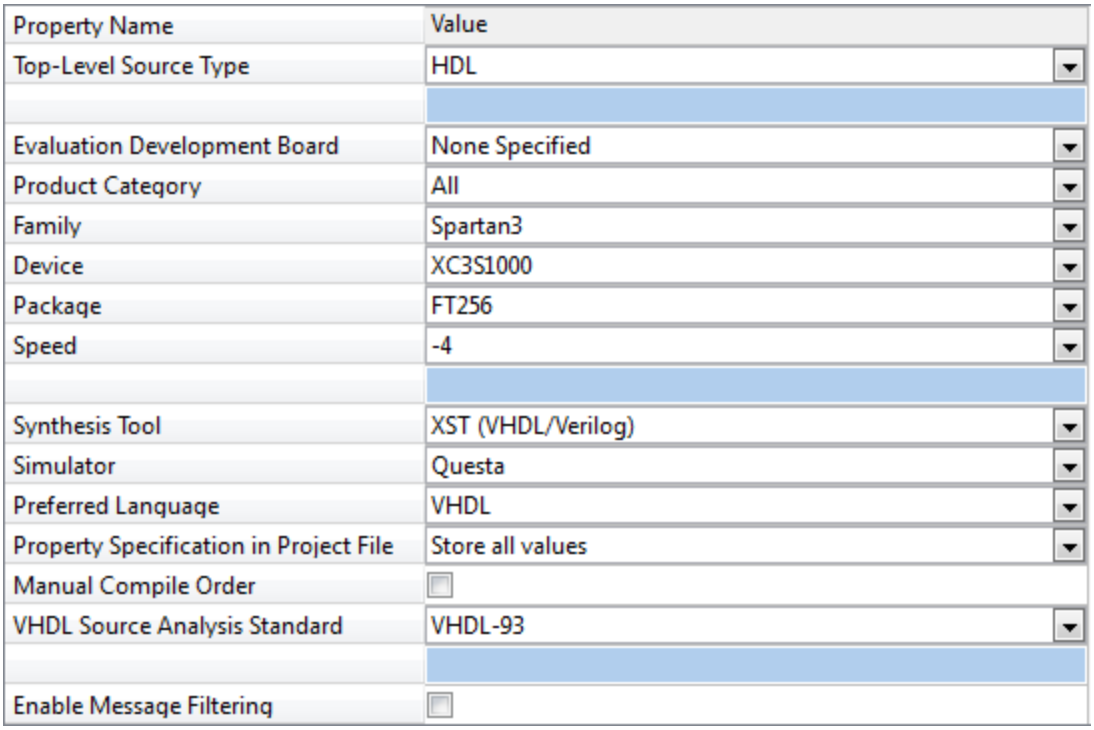

Рисунок 6.1.1 — Параметры синтеза

После того, как все несинтезируемые конструкции были заменены на синтезируемые, были получены результаты для матрицы размерностью 2x2, подробнее на рисунке 6.1.2.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Latches	307	15,360	1%
Number of 4 input LUTs	7,947	15,360	51%
Number of occupied Slices	4,036	7,680	52%
Number of Slices containing only related logic	4,036	4,036	100%
Number of Slices containing unrelated logic	0	4,036	0%
Total Number of 4 input LUTs	7,954	15,360	51%
Number used as logic	7,947		
Number used as a route-thru	7		
Number of bonded IOBs	9	173	5%
IOB Flip Flops	4		
Number of MULT18X18s	1	24	4%
Number of BUFGMUXs	2	8	25%
Average Fanout of Non-Clock Nets	4.86		

Рисунок 6.1.2 — Результаты синтеза для матрицы 2x2

6.2 Экспериментальное исследование зависимости увеличения аппаратной сложности от увеличения числа аргументов функции

Для Spartan3 – 1000 данная задача оказалась практически нереализуемой. Переполнение произошло при размерности матрицы 4x4 (рисунок 6.2.1), а это очень мало. Решением данной проблемы может быть использование другой базы, либо использование последовательного ввода и расчета координат, но это негативно скажется на времени работы программы.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Latches	670	15,360	4%	
Number of 4 input LUTs	20,093	15,360	130%	OVERMAPPED
Number of occupied Slices	10,100	7,680	131%	OVERMAPPED
Number of Slices containing only related logic	10,100	10,100	100%	
Number of Slices containing unrelated logic	0	10,100	0%	
Total Number of 4 input LUTs	20,199	15,360	131%	OVERMAPPED
Number used as logic	20,093			
Number used as a route-thru	106			
Number of bonded IOBs	23	173	13%	
IOB Flip Flops	6			
Number of MULT18X18s	1	24	4%	
Number of BUFGMUXs	2	8	25%	
Average Fanout of Non-Clock Nets	4.90			

Рисунок 6.2.1 — Результаты синтеза для матрицы 4x4

Аппаратная сложность устройства зависит лишь от одного параметра N – размерность матрицы. График зависимости роста аппаратной сложности от размерности матрицы представлен на графике 6.2.1. Конкретные значения графика приведены в таблице 6.2.1.

Таблица 6.2.1 — Зависимость заполненности ПЛИС от N

N	Latches	LUTs	Slices	IOBs	MULT 18X18s	BUDGMUXs
2	307	7954	4036	9	1	2
3	446	14604	7388	16	1	2
4	670	20199	10100	23	1	2
5	923	22841	11421	32	1	2
10	2590	61597	30865	109	1	2

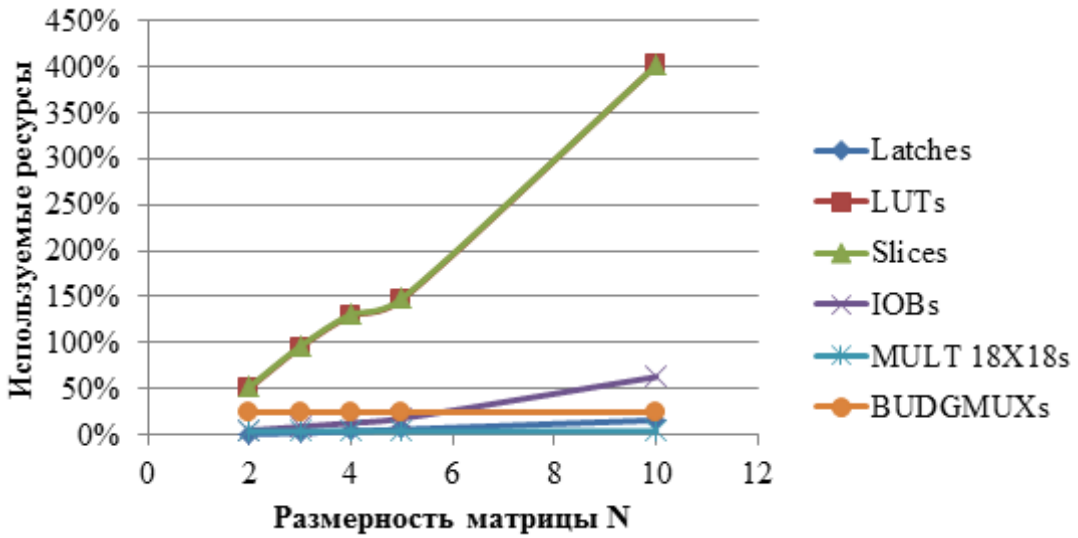

График 6.2.1 — Рост аппаратной сложности

Из графика видно, что первыми переполняются слайсы из-за увеличения числа лутов. Остальные параметры возрастают достаточно медленно.

ЗАКЛЮЧЕНИЕ

Данная работа, бесспорно, крайне важна. Она позволяет закрепить те базовые моменты, которые крайне важны для каждого инженера. Максимальная приближенность курсового проекта к реальным условиям работы позволяет ощутить себя на месте специалистов, перед которыми поставлена конкретная задача.

Крайне интересно было не только опытным путем определить наилучший способ реализации алгоритма, но и добиться синтезируемости проекта. Было проведено множество попыток и проверок результатов при различных реализациях проекта. К сожалению, ПЛИС фирмы Xilinx семейства Spartan3-1000 не имеет достаточно ресурсов для выполнения данной задачи. Однако на более мощных ПЛИС данный алгоритм возможно реализовать для более приемлемых размерностей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бибило П.Н. Основы языка VHDL; Изд. 3-е, доп. - М.: Издательство ЛКИ, 2007. - 328 с.
2. Бибило П.Н. Задачи по проектированию логических схем с использованием языка VHDL; Издательство ЛКИ, 2010г. – 328 с.
3. Царев Ф.Н., Шалыто А.А. Применение генетического программирования для построения мультиагентной системы одного класса / Международная научнотехническая мультиконференция «Проблемы информационно-компьютерных технологий и мехатроники». Материалы международной научно-технической конференции «Многопроцессорные вычислительные и управляющие системы» (МВУС`2007). – Таганрог: НИИМВС, 2007. – Т.2. – С. 46–51.
4. Брауэр В. Введение в теорию конечных автоматов. – М.: Радио и связь, 1987.
5. Красс А. Метод обучения сложных систем с большим числом сенсоров и актуаторов. Бакалаврская работа / СПбГУ ИТМО, факультет «Информационные технологии и программирование», кафедра «Компьютерные технологии». – 2008.
6. Царев Ф.Н., Шалыто А.А. О построении автоматов с минимальным числом состояний для задачи об «умном муравье» / Сборник докладов X международной конференции по мягким вычислениям и измерениям. – СПбГЭТУ "ЛЭТИ". – Т. 2. – 2007. – С. 88–91.