Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ
И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных средств

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту на тему:

VHDL - МОДЕЛЬ МИКРОПРОГРАММНОГО АВТОМАТА УМНОЖЕНИЯ ЧИСЕЛ

БГУИР КП 1-40 02 02 011 ПЗ

Студент А.В. Миронь
Руководитель П.Н. Бибило

Минск 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Язык VHDL (Very high speed integrated circuits Hardware Description Language) был разработан в США по инициативе министерства обороны в 1987 году. VHDL был принят в качестве стандарта ANSI/IEEE Std 1076-1987. Затем язык был усовершенствован, новый стандарт ANSI/IEEE Std 1076-1993 (стандарт VHDL'93) появился в 1993 году.

VHDL – это мощный язык, который предназначен для описания проектов различной степени сложности – от простейшего вентиля до целой системы, состоящей из аппаратных и программных частей. Он позволяет описывать поведение, т.е. алгоритмы функционирования цифровых систем; строить модели на различных уровнях абстракции, выполнять имитационное моделирование и генерировать временные диаграммы, вести строгое документирование проекта, осуществлять синтез структуры по поведенческому описанию, автоматически генерировать тесты, а также проводить иерархическое функционально-структурное описание систем; имеет средства для описания параллельных асинхронных процессов, регулярных (систолических) структур и в то же время имеет все признаки языка программирования высокого уровня - позволяет создавать свои типы данных, имеет широкий набор арифметических и логических операций и т.д. Использование этого языка позволяет не привязывать проект заранее к конкретному физическому способу реализации.

Язык VHDL развивается, ему посвящаются международные конференции, выходят научные журналы, в которых изучаются проблемы использования VHDL. Он стал языком разработки международных проектов, в том числе осуществляемых с помощью всемирной компьютерной сети Internet. Знакомство с этим языком необходимо для эффективной работы по созданию самой разнообразной электронной аппаратуры на современной элементной базе сверхбольших интегральных схем.

1 РАЗРАБОТКА АЛГОРИТМА

1.1 Постановка задачи

Задано два двоичных числа A и B – векторы размерностью n = 4, …, 10 элементов. Необходимо построить микропрограммный автомат для умножения двоичных чисел. Будет использоваться три регистра и сумматор: RG1 – регистр, в котором постоянно хранится значение множимого A; RG2 – регистр, в котором хранится множитель B; RG3 – регистр, в котором размещаются частичные суммы частичных произведений и произведение. Умножение будет производиться, начиная с младших разрядов, тем самым формируя частичные произведения.

1.2 Краткие теоретические сведения

Производительность ЭВМ в значительной степени определяется временем выполнения операции умножения. Программная реализация операции умножения двух операндов A и B с помощью операций сдвига и сложения требует значительного времени.

Простейший способ построения умножителя n × m разрядов следует из алгоритма умножения двоичных чисел:

$$P = A \times B = A \times (b_{m-1}2^{m-1} + \ldots + b_1 2^1 + b_0 2^0) = \sum_{j=0}^{m-1} A \times b_j 2^j = \sum_{j=0}^{m-1} P_j, \quad (1.1)$$

где $A = a_{n-1} \ldots a_1 a_0$ – множимое, $B = b_{m-1} \ldots b_1 b_0$ – множитель, $P_j = A \times b_j 2^j$ – частичные произведения.

1.3 Алгоритм

Для обоснования принципа построения микропрограммного автомата необходимо воспользоваться правилом умножения двоичных чисел, при котором частичные произведения формируются, начиная с младших разрядов множителя B (рисунок 1.1).

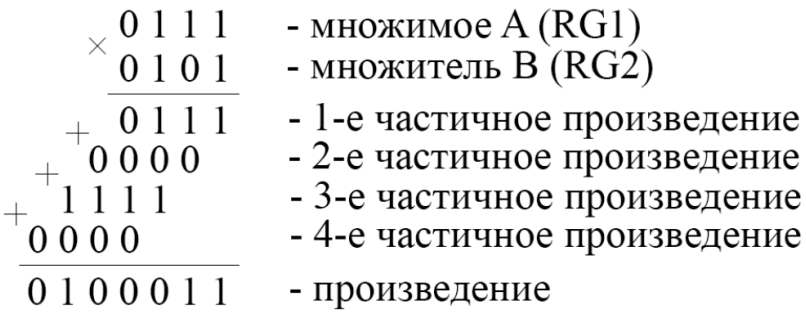


Рисунок 1.1 – Правило умножения двоичных чисел

Суммирование частичных произведений должно осуществляться последовательно во времени с помощью сумматоров, предназначенных для сложения двух операндов. Особенность умножения двоичных чисел состоит в том, что частичные произведения могут принимать лишь два значения: значение множимого A либо значение нуля. Значение частичного произведения определяется значением текущего разряда множителя B. Если частичное произведение равно нулю, то микрооперацию сложения можно не выполнять.

Множимое A используется как частичное произведение, и оно будет храниться в регистре RG1. Для определения значения текущего разряда множителя B необходимо иметь сдвигающий регистр RG2. В исходном состоянии регистр RG2 загружен множителем B. Чтобы выявить значение следующего разряда множителя B, после каждой микрооперации сложения частичного произведения необходимо производить сдвиг содержимого RG2 в сторону самого младшего разряда. Для хранения частичных сумм частичных произведений необходимо располагать третьим регистром RG3. В исходном состоянии RG3 должен быть загружен нулями. В процессе умножения осуществляется сложение содержимого регистра RG3 с частичным произведением A. Частичная сумма помещается в RG3, после чего выполняется сдвиг в сторону младших разрядов. При этом в два раза увеличивается вес каждого очередного разряда множителя B.

На рисунке 1.2 показан процесс умножения с использованием трех регистров и сумматора. Множимое A = 0111 постоянно находится в регистре RG1. В исходном состоянии в регистр RG3 помещен нуль 0000, а в регистр RG2 – множитель B = 0101. Нуль в старшем разряде операндов A и B свидетельствует о том, что перемножаются положительные числа. В процессе умножения в регистре RG3 размещаются частичные суммы частичных произведений и произведение.

Рисунок 1.2 – Алгоритм умножения

Этапы умножения:

1. Анализируется младший разряд регистра RG2:

- Если младший разряд RG2 равен 1, то выполняется микрооперация сложения содержимого регистров RG3 и RG1, и результат помещается в RG3.

- Если младший разряд RG2 равен 0, то переходим к этапу 2.

2. Выполняется микрооперация сдвига вправо на один разряд содержимого составного регистра (RG3 и RG2).

Этот процесс носит циклический характер. Число циклов n равно числу разрядов множителя A (в данном случае n = 4).

2 РАЗРАБОТКА VHDL-МОДЕЛИ

Загрузка исходных данных производится из текстового файла (рисунок 2.1). Двоичные числа разделены символом «пробел». Каждая новая пара множимого и множителя записывается с новой строки. Входные данные параметризируются в пакете «project_package». Пакет представлен в листинге 2.1.

```
0111 0000
0111 0001
0111 0010
0111 0011
0111 0100
0111 0101
0111 0110
0111 0111
```

Рисунок 2.1 – Представление исходных данных в текстовом файле

2.1 Пакеты

Константы и типы описаны в пакете project_package.

Листинг 2.1 – Пакет «project_package»

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

PACKAGE project_package IS
    CONSTANT N : NATURAL := 4;
    TYPE int_array IS ARRAY(NATURAL RANGE 0 TO N - 1) OF INTEGER
RANGE 0 TO 3;
END project_package;
```

2.2 Entity «multiplier» и ее архитектура

Листинг 2.2 – Код умножителя

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY work;
USE work.project_package.ALL;

ENTITY multiplier IS
     PORT (
          rg1, rg2 : IN std_logic_vector (0 TO N - 1);
```

```
          result : OUT std_logic_vector (0 TO (N * 2) - 1));
END multiplier;

ARCHITECTURE mult_function_beh OF multiplier IS
     FUNCTION  fill_array_with_zeros(arr  :  int_array)  RETURN
int_array IS
          VARIABLE arr_with_zeros : int_array;
     BEGIN
          FOR i IN 0 TO arr'length - 1 LOOP
               arr_with_zeros(i) := 0;
          END LOOP;
          RETURN arr_with_zeros;
     END FUNCTION;

     FUNCTION std_logic_to_integer(arr : std_logic_vector(0 TO N
- 1)) RETURN int_array IS
          VARIABLE arr_to_integer : int_array;
     BEGIN
          FOR i IN 0 TO arr'length - 1 LOOP
               IF (arr(i) = '1') THEN
                    arr_to_integer(i) := 1;
               ELSE
                    arr_to_integer(i) := 0;
               END IF;
          END LOOP;

          RETURN arr_to_integer;
     END FUNCTION;

     FUNCTION integer_to_std_logic(arr1, arr2 : int_array) RETURN
std_logic_vector IS
          VARIABLE arr_to_std_logic : std_logic_vector(0 TO (N *
2) - 1);
     BEGIN
          FOR i IN 0 TO arr1'length - 1 LOOP
               IF (arr1(i) = 1) THEN
                    arr_to_std_logic(i) := '1';
               ELSE
                    arr_to_std_logic(i) := '0';
               END IF;
          END LOOP;

          FOR i IN 0 TO arr2'length - 1 LOOP
               IF (arr2(i) = 1) THEN
                    arr_to_std_logic(i + N) := '1';
               ELSE
```

```
                    arr_to_std_logic(i + N) := '0';
               END IF;
          END LOOP;

          RETURN arr_to_std_logic;
     END FUNCTION;

     FUNCTION multiply(rg1, rg2 : std_logic_vector(0 TO N - 1))
RETURN std_logic_vector IS
          VARIABLE rg1_copy : int_array;
          VARIABLE rg2_copy : int_array;
          VARIABLE rg3 : int_array;
          VARIABLE result : std_logic_vector (0 TO (N * 2) - 1);
          VARIABLE counter : INTEGER := N;
          VARIABLE buff : INTEGER := 0;
          VARIABLE rg3_last_element : INTEGER := 0;

     BEGIN
          rg1_copy := std_logic_to_integer(rg1);
          rg2_copy := std_logic_to_integer(rg2);
          rg3 := fill_array_with_zeros(rg3);

          WHILE counter > 0 LOOP
               IF (rg2_copy(N - 1) = 1) THEN
                    FOR i IN rg3'length - 1 DOWNTO 0 LOOP
                         rg3(i) := rg3(i) + rg1_copy(i);

                         IF (buff /= 0) THEN
                              rg3(i) := rg3(i) + buff;
                              buff := buff - 1;
                         END IF;
                         IF (rg3(i) > 1) THEN
                              buff := buff + 1;
                              rg3(i) := rg3(i) - (buff + 1);
                         END IF;
                    END LOOP;
               END IF;

               rg3_last_element := rg3(N - 1);

               FOR i IN rg3'length - 1 DOWNTO 1 LOOP
                    rg3(i) := rg3(i - 1);
               END LOOP;
               rg3(0) := 0;

               FOR i IN rg2_copy'length - 1 DOWNTO 1 LOOP
```

```
                    rg2_copy(i) := rg2_copy(i - 1);
                END LOOP;
                rg2_copy(0) := rg3_last_element;

                counter := counter - 1;
            END LOOP;

            result := integer_to_std_logic(rg3, rg2_copy);

            RETURN result;
        END multiply;

BEGIN
        result <= multiply(rg1, rg2);
END mult_function_beh;
```

3 РАЗРАБОТКА ТЕСТИРУЮЩЕЙ ПРОГРАММЫ

Для тестирования умножителя двоичных чисел была написана VHDL-программа, в которой множимое и множитель считываются из текстового файла каждые 50 нс.

Листинг 3.1 – Тестирующая программа

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_textio.ALL;
USE ieee.numeric_std.ALL;
USE std.textio.ALL;

LIBRARY work;
USE work.project_package.ALL;

ENTITY test_mult IS
END test_mult;

ARCHITECTURE test_beh OF test_mult IS
COMPONENT multiplier IS
     PORT (
          rg1, rg2 : IN std_logic_vector (0 TO N - 1);
          result : OUT std_logic_vector (0 TO (N * 2) - 1));
END COMPONENT;

SIGNAL rg1, rg2 : std_logic_vector (0 TO N - 1);
SIGNAL result : std_logic_vector (0 TO (N * 2) - 1);

CONSTANT f_name : STRING := "input_data.txt";
FILE f_source : text;

BEGIN

mult_func_call : multiplier PORT MAP(rg1 => rg1, rg2 => rg2, result
=> result);

PROCESS
     VARIABLE f_line : line;
     VARIABLE rg1_line : std_logic_vector(0 TO N - 1);
     VARIABLE rg2_line : std_logic_vector(0 TO N - 1);
     VARIABLE space_char : CHARACTER;
```

```
BEGIN
     file_open(f_source, f_name, read_mode);

     WHILE (NOT endfile(f_source)) LOOP
          readline(f_source, f_line);
          read(f_line, rg1_line);
          read(f_line, space_char);
          read(f_line, rg2_line);

          rg1 <= rg1_line;
          rg2 <= rg2_line;

          WAIT FOR 50 ns;
     END LOOP;

     file_close(f_source);

     END PROCESS;
END test_beh;
```

4 МОДЕЛИРОВАНИЕ И ОТЛАДКА VHDL ПРОГРАММ

Моделирование цифровых систем является важным этапом в процессе разработки. Целями моделирования могут быть как исследование алгоритмов работы проектируемого устройства, так и верификация характеристик, получаемых при его аппаратной реализации.

При моделировании используется подход, основанный на «испытательном стенде». Моделируемое устройство представляется своим синтезируемым кодом, а для проверки его поведения создаются описания тестовых воздействий.

Граф-схема устройства умножения представлена в приложении А.

Схемы операционного и управляющего автоматов представлены в приложении Б.

Временная диаграмма представлена в приложении В.

4.1 Моделирование верхнего модуля

Моделирование и верификация проводилась в системе ModelSim с помощью тестирующей программы, приведённой в листинге 3.1. В результате работы не было выведено сообщений об ошибках в тестирующей модели, следовательно, VHDL-модель устройства можно считать эквивалентной эталонной модели и реализованной корректно.

4.2 Покрытие VHDL-кода

Покрытие характеризует текст программы с точки зрения прохождения VHDL кода при моделировании. Применение процедур покрытия кода не предназначена для проверки правильности ожидаемых и полученных реакций VHDL-модели цифровой системы на наборах значений входных сигналов. Если промоделированная строка была выполнена хотя бы один раз, то она считается покрытой, иначе строка непокрыта. Результат покрытия VHDL-кода в среде ModelSim моделируемого устройства продемонстрирован на рисунке 4.1.

Coverage Summary by Structure:

Design Scope	Hits %	Coverage %
test_mult	92.45%	91.66%
mult_func_call	91.48%	91.66%

Coverage Summary by Type:

Total Coverage:					92.45%	91.66%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	62	62	0	1	100.00%	**100.00%**
Branches	12	12	0	1	100.00%	**100.00%**
Toggles	32	24	8	1	75.00%	**75.00%**

Рисунок 4.1 – Отчет о покрытии тестами в HTML формате

5 РАЗРАБОТКА ПРОГРАММ ВЕРИФИКАЦИИ

Для реализации метода, описанного в пункте 1, была разработана верификационная программа на языке Java. Код программы представлен в листингах 5.1-5.2, основная функция приведена в листинге 5.2. Результат выполнения программы представлен в приложении Г.

Листинг 5.1 – Ввод данных

```
private static final int NUMBER_LENGTH = 4;
private static final Scanner scanner = new Scanner(System.in);

public void start() {
    int[] rg1 = new int[NUMBER_LENGTH];
    int[] rg2 = new int[NUMBER_LENGTH];

    System.out.print("\nEnter the 1st number: ");
    enterNumber(rg1);
    System.out.print("\nEnter the 2nd number: ");
    enterNumber(rg2);
}
```

Листинг 5.2 – Умножение чисел

```
private void multiply(int[] rg1, int[] rg2) {
    int[] rg3 = new int[NUMBER_LENGTH];
    int[] result = new int[NUMBER_LENGTH * 2];
    int counter = rg1.length;
    int buffer = 0;

    while (counter > 0) {
        if (rg2[NUMBER_LENGTH - 1] == 1) {
            for (int i = rg3.length - 1; i >= 0; i--) {
                rg3[i] += rg1[i];
                if (buffer != 0) {
                    rg3[i] += buffer;
                    buffer--;
                }
                if (rg3[i] > 1) {
                    buffer++;
                    rg3[i] -= (buffer + 1);
                }
            }
        }
```

```
            int rg3LastElement = rg3[NUMBER_LENGTH - 1];
            for (int i = rg3.length - 1; i > 0; i--) {
                rg3[i] = rg3[i - 1];
            }
            rg3[0] = 0;

            for (int i = rg2.length - 1; i > 0; i--) {
                rg2[i] = rg2[i - 1];
            }
            rg2[0] = rg3LastElement;

            counter--;
        }

        for (int i = 0; i < rg3.length; i++) {
            result[i] = rg3[i];
        }
        for (int i = 0; i < rg2.length; i++) {
            result[i + NUMBER_LENGTH] = rg2[i];
        }
    }
```

6 СИНТЕЗ УСТРОЙСТВА НА ЭЛЕМЕНТНОЙ БАЗЕ ПЛИС

Этап синтеза представляет собой процесс трансформации исходного HDL-описания проектируемого устройства в список цепей, выполненный на низком логическом уровне. Элементы низкоуровневого описания, формируемого в процессе синтеза, должны соответствовать архитектуре семейства ПЛИС, выбранного для реализации проекта.

В синтезе выделяют следующие этапы:

– Высокоуровневый синтез

– Технологически независимые оптимизации

– Технологическое отображение

– Увеличение быстродействия

На этапе высокоуровневого синтеза осуществляется замена HDL конструкций соответствующими подсхемами. Во время этапа технологически независимых оптимизаций осуществляется оптимизация комбинационной логики. На третьем этапе осуществляется покрытие минимизированных логических выражений описаниями логических элементов, алгоритмическое описание триггеров заменяется описанием триггеров в целевой библиотеке. На заключающем этапе полученное структурное описание модифицируется, выделяются критические пути, происходит повторный синтез и быстродействие схемы увеличивается.

Для синтеза была использована программа Leonardo 2010a.7.

6.1 Описание целевой ПЛИС

Количество элементов платы Spartan 3 XC3S50:

- Slices: 768;
- Slice Flip Flops: 1536;
- 4 input LUTs: 1536;
- Bonded IOBs: 124;
- GCLKs: 8.

7 ОЦЕНКА АППАРАТНОЙ СЛОЖНОСТИ

Оценка аппаратной сложности производилась путем изменения количества разрядов двоичных чисел. График зависимости используемой аппаратуры продемонстрирован на рисунке 7.1.

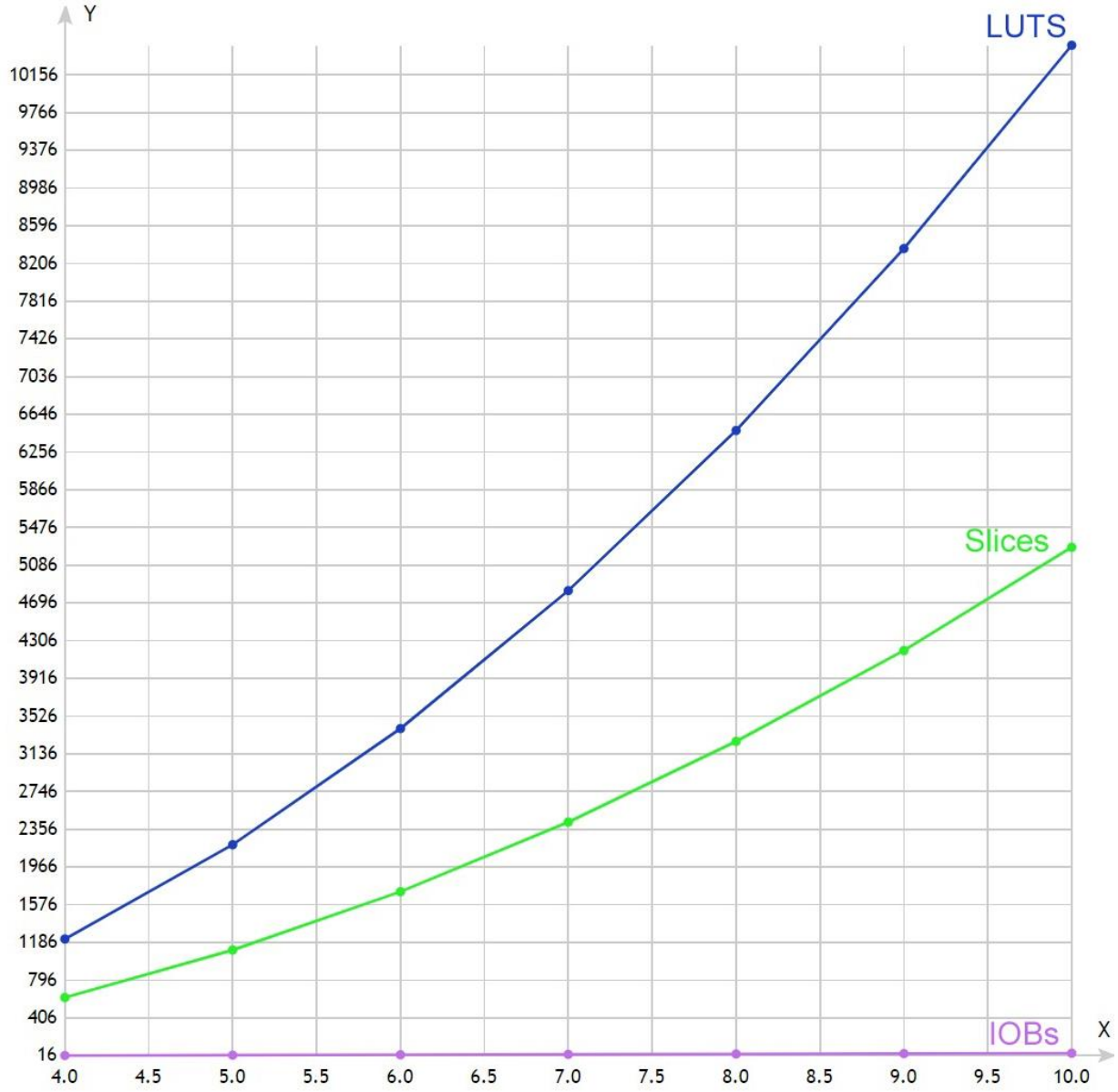


Рисунок 7.1 – Оценка аппаратной сложности

8 ЗАКЛЮЧЕНИЕ

В данном курсовом проекте была разработана VHDL-модель микропрограммного автомата умножения чисел.

В ПЛИС Spartan-3 XC3S1000 помещается проектируемое устройство, выполняющее умножение двух двоичных чисел с количеством от 4 до 10 разрядов. Выполнение курсового проекта максимально приближено к промышленной разработке цифровых устройств что позволило более глубоко изучить и систематизировать знания в области проектирования цифровых средств на языках описания аппаратуры.

Схемная реализация поставленной задачи на ПЛИС имеет большой объём, что показывает важность использования языков описания аппаратуры таких, как VHDL, чтобы значительно ускорить процесс разработки, а также сделать его более похожим на написание прикладных программ на языке высокого уровня.

9 СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

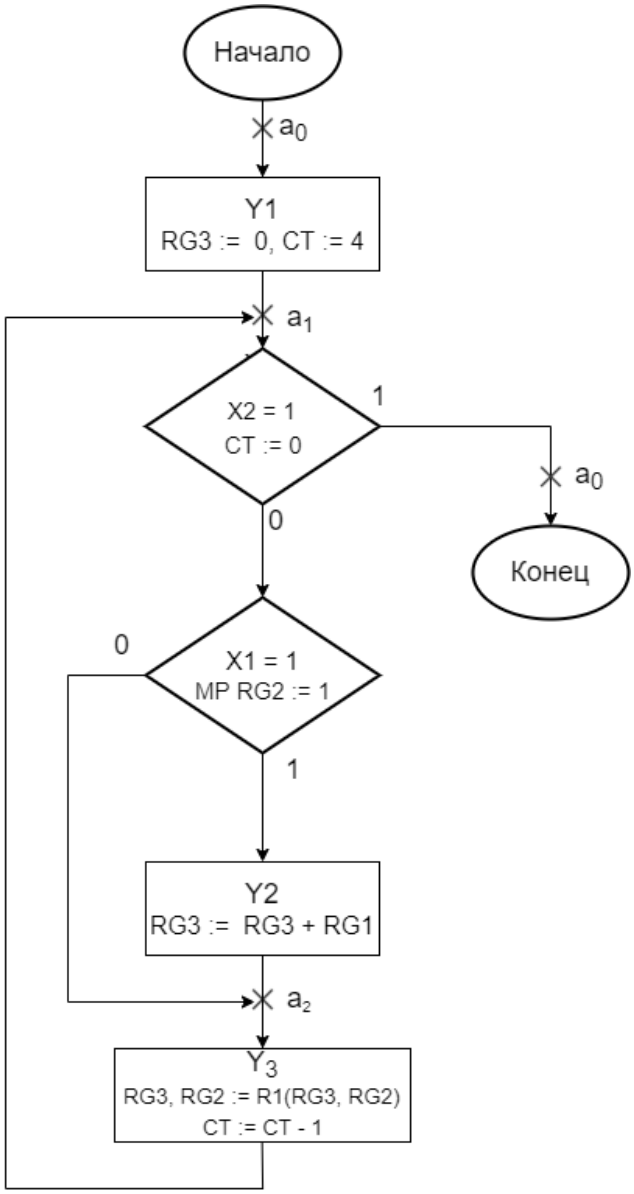
[1] Бобрешов А.М. Проектирование цифровых устройств с помощью языка описания аппаратуры VHDL: учебное пособие/Бобрешов А.М., Дыбой А.В. - Воронеж: ИПЦ ВГУ, 2007. - 51 с.

[2] Бибило П.Н. Основы языка VHDL: учебное пособие/Бибило П.Н. – Москва, «Солон-Р», 2010. – 200 с.

[3] Бибило П.Н., Авдеев Н.А. VHDL. Эффективное использование при проектировании цифровых систем – Москва, «Солон-Р», 2006. – 344 с.

10 ПРИЛОЖЕНИЕ А

Граф-схема устройства умножения.



11 ПРИЛОЖЕНИЕ Б

Схема операционного автомата.

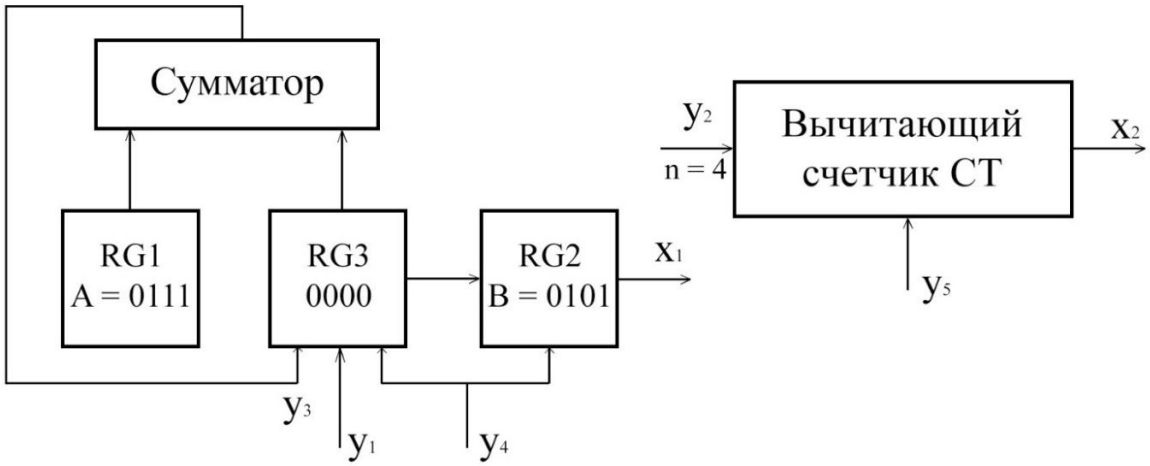


Схема управляющего автомата.

12 ПРИЛОЖЕНИЕ В

Временная диаграмма представлена для примера из пункта 1.

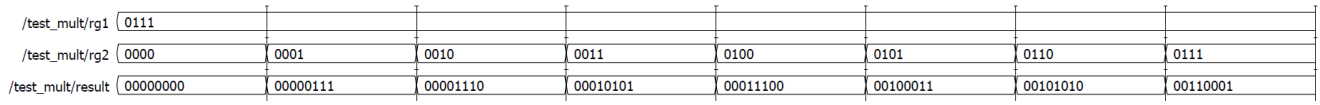


Диаграмма представлена для времени моделирования от 2800 до 3200 нс.

RG1 – множимое A.

RG2 – множитель B.

Result – результат произведения.

13 ПРИЛОЖЕНИЕ Г

Результат выполнения верификационной программы по исходным данным примера из пункта 1.

```
Enter the 1st number: 0111
Enter the 2nd number: 0101

The result of multiplication: 00100011
```