



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

Институт информационных технологий  
Кафедра вычислительной техники

## КУРСОВАЯ РАБОТА

по дисциплине Схемотехника устройств компьютерных систем  
(наименование дисциплины)

Тема курсовой работы Проектирование устройства для минимизации логической  
функции методом Куайна-Мак-Класски

Студент группы ИББО-04-21 Паращенко Федор Дмитриевич  
(учебная группа, фамилия, имя отчество, студента)

  
(подпись студента)

Руководитель курсовой  
работы

профессор, д.т.н., Потехин Д.С.  
(должность, звание, ученая степень)

  
(подпись руководителя)


Консультант

преподаватель, Люлява Д.В.  
(должность, звание, ученая степень)

  
(подпись консультанта)

Работа представлена к защите « 23 » декабря 2023г.

Допущен к защите « 23 » декабря 2023г.

ОТЛИЧНО 

Москва 2023 г.



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий

Кафедра вычислительной техники

Утверждаю

Заведующий кафедрой

Платонова О.В.

ФИО

«15» сентября 2023 г.

### ЗАДАНИЕ

на выполнение курсовой работы по дисциплине  
«Схемотехника устройств компьютерных систем»

Студент Паращенко Федор Дмитриевич Группа ИБВО-04-21

Тема: Проектирование устройства для минимизации логической  
функции методом Куайна-Мак-Класски

Исходные данные: ПЛИС XC7A100TCSG324-1L, маршрут проектирования СБИС, Verilog HDL

Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Провести анализ предметной области.
2. Разработать набор модулей, описывающих процесс минимизации функции методом Куайна-Мак-Класски.
3. Разработать набор модулей, описывающих процессы приёма и обработки данных по протоколу UART.
4. Разработать модуль верхнего уровня, реализующий управление для совместной работы всех модулей.
5. Провести верификацию.
6. Составить отчетную документацию по проделанной работе.

Срок представления к защите курсовой работы:

до «28» декабря 2023г.

Задание на курсовую работу выдал

Подпись руководителя

( Потехин Д.С. )  
Ф.И.О. руководителя

Задание на курсовую работу получил

Подпись обучающегося

«15» сентября 2023г.

( Паращенко Ф.Д. )  
Ф.И.О. исполнителя

Москва 2023 г.

## **АННОТАЦИЯ**

Данная работа включает в себя 10 рисунков, 23 листинга, и 13 приложений. Количество страниц в работе — 85.

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	6
1 СИСТЕМНАЯ МОДЕЛЬ .....	7
1.1 Описание предметной области .....	7
1.1.1 Минимизация методом Куайна-Мак-Класки .....	7
1.1.2 Создание устройств на базе ПЛИС .....	8
1.2 Реализация системной модели .....	8
1.3 Тестирование .....	11
2 RTL-МОДЕЛЬ УСТРОЙСТВА .....	15
2.1 Структурная схема RTL-модели .....	15
2.2 Описание модулей RTL-модели .....	16
2.3 Верификация RTL-модели .....	19
3 РАЗМЕЩЕНИЕ НА ПЛИС .....	23
3.1 Основные подходы к размещению .....	23
3.2 Набор ограничений .....	25
3.3 Вариант размещения .....	26
4 ТЕСТИРОВАНИЕ .....	28
ЗАКЛЮЧЕНИЕ .....	32
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	33
ПРИЛОЖЕНИЯ .....	35
Приложение А.1 .....	36
Приложение А.2 .....	42
Приложение Б.1 .....	45
Приложение Б.2 .....	48

Приложение Б.3 .....	49
Приложение Б.4 .....	63
Приложение Б.5 .....	67
Приложение Б.6 .....	69
Приложение Б.7 .....	70
Приложение Б.8 .....	72
Приложение Б.9 .....	74
Приложение Б.10 .....	75
Приложение Б.11 .....	76
Приложение В.....	77

## ВВЕДЕНИЕ

Целью курсовой работы является проектирование устройства для минимизации логической функции методом Куайна-Мак-Класски.

Предметная область: «Создание устройств на базе ПЛИС», «Минимизация методом Куайна-Мак-Класски».

Для проектирования заданного устройства необходимо выполнить следующие шаги:

1. Провести анализ предметной области.
2. Разработать набор модулей, описывающих процесс минимизации функции методом Куайна-Мак-Класски.
3. Разработать набор модулей, описывающих процессы приёма и обработки данных по протоколу UART.
4. Разработать модуль верхнего уровня, реализующий управление для совместной работы всех модулей.
5. Провести верификацию.
6. Составить отчетную документацию по проделанной работе.

Проектирование вычислительного устройства на базе ПЛИС (программируемых логических интегральных схемах) для минимизации логической функции методом Куайна-Мак-Класски является актуальной задачей в области цифровой электроники и компьютерных наук. Этот метод используется для упрощения булевых выражений, что может быть полезно в проектировании цифровых схем и микропроцессоров.

# 1 СИСТЕМНАЯ МОДЕЛЬ

## 1.1 Описание предметной области

### 1.1.1 Минимизация методом Куайна-Мак-Класки

Метод Куайна-Мак-Класки представляет собой табличный метод минимизации булевых функций, предложенный Уиллардом Куайном и усовершенствованный Эдвардом Мак-Класки. Этот метод является попыткой избавиться от недостатков метода Куайна [14].

Алгоритм минимизации метода Куайна-Мак-Класки включает следующие шаги:

Термы (конъюнктивные в случае СДНФ и дизъюнктивные в случае СКНФ), на которых определена функция алгебры логики записываются в виде их двоичных эквивалентов.

Эти эквиваленты разбиваются на группы, в каждую группу входят эквиваленты с равным количеством единиц (нулей).

Производится попарное сравнение эквивалентов (термов) в соседних группах, с целью формирования термов более низких рангов.

Составляется таблица, заголовком строк в которой являются исходные термы, а заголовком столбцов — термы низких рангов.

Расставляются метки, отражающие поглощение термов высших рангов (исходных термов) и далее минимизация производится по методу Куайна.

Особенностью метода Куайна-Мак-Класки по сравнению с методом Куайна в сокращении количества попарных сравнений на предмет их склеивания. Сокращение достигается за счёт исходного разбиения термов на группы с равным количеством единиц (нулей). Это позволяет исключить сравнения, заведомо не дающие склеивания.

Метод Куайна-Мак-Класки также имеет ограничения области применения, так как время работы метода растёт экспоненциально с

увеличением входных данных. Для функции от  $n$  переменных верхняя граница количества основных импликант  $3^n/n$ . Если  $n = 32$  их может быть больше чем  $6.5 \cdot 10^{12}$ .

### **1.1.2 Создание устройств на базе ПЛИС**

ПЛИС — это электронные компоненты, используемые для создания конфигурируемых цифровых электронных схем. В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задается посредством программирования. Для программирования используются программатор и программное обеспечение САПР, позволяющие задать желаемую структуру цифрового устройства в виде принципиальной электрической схемы или программы на специальных языках описания аппаратуры, таких как Verilog, VHDL, AHDL и других.

ПЛИС широко используются для построения различных по сложности и по возможностям цифровых устройств, включая устройства с большим количеством портов ввода-вывода, устройства, выполняющие передачу данных на высокой скорости, устройства, выполняющие криптографические операции, системы защиты информации, устройства, предназначенные для проектирования и прототипирования интегральных схем специального назначения (ASIC), устройства, выполняющие роль мостов (коммутаторов) между системами с различной логикой и напряжением питания.

## **1.2 Реализация системной модели**

Для реализации системной модели был выбран язык программирования Java. Была разработана программа для минимизации булевой функции в векторном виде.

На вход программа принимает разрядность функции, затем саму функцию. Результатом выполнения является минимизированная функция в буквенном формате. Литерал в нижнем регистре обозначает инверсию данного



разряда, литерал в верхнем регистре означает логическую единицу в данном разряде. На Рисунке 1.1 представлен вывод программы.

```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" ...
[1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 3, 1, 0, 1, 4] [0, 0, 0, 3, 3, 4] [3, 0, 3, 0, 0, 4] [3, 0, 0, 3, 1, 4] [0, 3, 0, 1, 3, 4] [1, 3, 1, 1, 3, 4] [1, 1, 3, 3, 3, 4]
[aCdE, abc, bde, bcE, acD, ACD, AB]

Process finished with exit code 0
```

**Рисунок 1.1 — Вывод системной модели**

На данном рисунке первая строка это заданный вектор, вторая строка вывод результата в цифровом формате, третья строка вывод результата в буквенном формате.

Программа разделена на выполнение нескольких этапов:

1. Получение массива всех импликант.
2. Распределение импликант на группы по количеству единиц.
3. Склеивание групп в цикле.
4. Получение массива простых импликант.
5. Заполнение таблицы Квайна.
6. Поиск ядерных импилкант.
7. С помощью одной из вариаций метода петрика производится поиск оставшихся импликант.

В Листинге 1.1 представлен фрагмент кода в котором происходит склеивание импилкант.

*Листинг 1.1 — Склеивание*

```
while(cf) {
    cf = false;

    for (cmp_u = 0; cmp_u < n; cmp_u++) {
        cmp_d = (byte) (cmp_u + 1);
        for (int i = 0; i < cn[ml][cmp_u]; i++) {
            for (int j = 0; j < cn[ml][cmp_d]; j++) {
                //Функции сравнения вынести в виде task'ов
                cmp_out = compare_for_merging(groups[ml][cmp_u][i],
groups[ml][cmp_d][j], n); //больше 0 если можно склеить
                if (cmp_out >= 0) {
                    cf = true;
                    local = groups[ml][cmp_u][i].clone();
                    local[cmp_out] = 3;
                    local[n] = 0;

                    groups[ml][cmp_u][i][n] = 4;
                    groups[ml][cmp_d][j][n] = 4;
                    c = count_of_1(local, n);
                    wf = true;
                }
            }
        }
    }
}
```

### *Продолжение Листинга 1.1*

```
        for (int p = 0; p < cn[ml + 1][c]; p++) {
            if (compare_implicants(local, groups[ml + 1][c][p], n))
        {
                wf = false;
                break;
            }
        }

        if (wf) {
            groups[ml + 1][c][cn[ml + 1][c]] = local;
            cn[ml + 1][c]++;
        }
    }
}

ml++;
}
```

При каждом склеивании новая импликанта, с одним пропущенным разрядом записывается в новую таблицу, следующего этапа склеивания. В таблицу следующего уровня записываются только уникальные импликанты, для этого при каждом склеивании происходит проверка. Выполнение заканчивается, когда в таблице не было ни одного склеивания. Склеенные импликанты помечаются занесением значения в отдельный разряд.

В Листинге 1.2 представлен фрагмент кода в котором происходит поиск оставшихся необходимых импликант с помощью одной из вариаций метода Петрика.

### *Листинг 1.2 — Поиск оставшихся импликант*

```
if (clpt[0] > 0) {
    while (true) {
        for (int i = 0; i < cpi; i++) {
            for (int j = 0; j < clpt[1]; j++) {
                if (petrick_table[l][j][i] == 1) {
                    clc++;
                }
            }
            if (clc > max_clc) {
                max_clc = clc;
                pei = i;
            }
            clc = 0;
        }

        pi[pei][n] = 4;
        for (int i = 0; i < clpt[1]; i++) {
            if (petrick_table[l][i][pei] != 1) {
```

### *Продолжение Листинга 1.2*

```
        petrick_table[l + 1][clpt[l + 1]] =
petrick_table[l][i].clone();
        clpt[l + 1]++;
    }
}
if (clpt[l + 1] == 0) {
    break;
}
l++;
max_clc = 0;
}
}

ArrayList<byte[]> res = new ArrayList<>();
for (int i = 0; i < cpi; i++){
    if(pi[i][n] == 4) {
        res.add(pi[i]);
    }
}
return res;
}
```

В цикле происходит выбор импикант, перекрывающих как можно больше единичных точек. После каждой выбранной импиканты происходит запись значений в новую таблицу, в которой происходит выбор следующей импиканты, пока таблица не окажется пустой.

В Приложении А.1 представлен полный код системной модели.

## **1.3 Тестирование**

Для тестирования системной модели были созданы модульные тесты. Пример представлен в Листинге 1.3.

### *Листинг 1.3 — Пример тестирования*

```
import my_system_model.Wrapper2;
import org.junit.jupiter.api.Test;
import system_model.Wrapper1;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class TestIndividualValues {

    @Test
    public void testAll0(){
        assertEquals(Wrapper2.calculate("000000000000000000000000000000", 5),
"0");
        assertEquals(Wrapper2.calculate("0000000000000000", 4), "0");
        assertEquals(Wrapper2.calculate("000000000", 3), "0");
    }

    @Test
```

### *Продолжение Листинга 1.3*

```
        public void testAll1(){
            assertEquals(Wrapper2.calculate("111111111111111111111111111111", 5),
"1");
            assertEquals(Wrapper2.calculate("1111111111111111", 4), "1");
            assertEquals(Wrapper2.calculate("11111111", 3), "1");
        }

        @Test
        public void testOneLetter(){
            assertEquals(Wrapper2.calculate("11111111111111110000000000000000", 5),
"a");
            assertEquals(Wrapper2.calculate("1111111100000000", 4), "a");
            assertEquals(Wrapper2.calculate("00001111", 3), "A");

            assertEquals(Wrapper2.calculate("11110000111100000000000000000000", 5),
"ac");
            assertEquals(Wrapper2.calculate("0011001100110011", 4), "C");
            assertEquals(Wrapper2.calculate("00110011", 3), "B");
        }
    }
```

Так же для тестирования системной модели была взята другая реализация метода Куайна-Мак-Класски [16], для сравнения полученных результатов. В Листинге 1.4 представлен код, в котором создаются 10000 тысяч случайных векторов, затем каждый вектор минимизируется с помощью обеих реализаций и происходит сравнение длин результатов и проверка корректности минимизации. В Приложении А.2 приведена реализация проверок.

### *Листинг 1.4 — Тестирование системной модели*

```
import org.apache.commons.math3.random.RandomDataGenerator;
import my_system_model.Wrapper2;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import system_model.Wrapper1;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Scanner;

import static java.lang.Math.pow;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

public class MainTest {
    protected static final String TESTING_VECTORS =
"src/test/resources/testingVectors.txt";
    protected static final String SYSTEM_MODEL_RESULTS =
"src/test/resources/systemModelResults.txt";
    protected static final String MY_SYSTEM_MODEL_RESULTS =
"src/test/resources/mySystemModelResults.txt";
    private static final int capacity = 5;
    private static final int testValuesCount = 100;

    private static final CheckMinimizedFunction checkMinimizedFunction = new
CheckMinimizedFunction();
```

#### Продолжение Листинга 1.4

```
@BeforeAll
static void calculations(){
    try {
        createVectors(capacity, testValuesCount);
        getResults1();
        getResults2(capacity);
        checkMinimizedFunction.launch(capacity, testValuesCount);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}

@Test
public void testCountOfResults(){
    try {
        int i = 0;
        Scanner scanner = new Scanner(new File(SYSTEM_MODEL_RESULTS));
        while(scanner.hasNextLine()){
            scanner.nextLine();
            i++;
        }
        scanner.close();
        assertEquals(i, testValuesCount);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}

@Test
public void testCountOfResultsMy(){
    try {
        int i = 0;
        Scanner scanner = new Scanner(new File(MY_SYSTEM_MODEL_RESULTS));
        while(scanner.hasNextLine()){
            i++;
            scanner.nextLine();
        }
        scanner.close();
        assertEquals(i, testValuesCount);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}

@Test
public void testCountOfReadResults(){
    assertEquals(checkMinimizedFunction.getTestingValues1().size(),
testValuesCount);
}

@Test
public void testCountOfReadResultsMy(){
    assertEquals(checkMinimizedFunction.getTestingValues2().size(),
testValuesCount);
}

@Test
public void testCountOfReadInputVectors(){
    assertEquals(checkMinimizedFunction.getInputVectors().size(),
testValuesCount);
}
```

#### Продолжение Листинга 1.4

```
@Test
public void testCheckMinimization(){
    checkMinimizedFunction.testCheckMinimization();
}
@Test
public void testCheckMinimizationMy(){
    checkMinimizedFunction.testCheckMinimizationMy();
}
@Test
public void testMetricForMinimizedFunction() throws FileNotFoundException {
    assertTrue(checkMinimizedFunction.checkMetricOfResultFunction());
}
private static void createVectors(int capacity, int count) throws
FileNotFoundException {
    int length = (int)pow(2, capacity);
    StringBuilder vect0 = new StringBuilder("000000000");
    for (int i = 0; i < capacity - 3; i++)
        vect0.append(vect0);
    StringBuilder vect = new StringBuilder(vect0);
    String binaryString;
    PrintWriter out = new PrintWriter(TESTING_VECTORS);
    long testingVector;
    for (int i = 0; i < count; i++){
        testingVector = new RandomDataGenerator().nextLong(0L, (long)pow(2,
length)-1);
        binaryString = Long.toBinaryString(testingVector);
        vect.replace(length - binaryString.length(), length, binaryString);
        out.println(vect);
        vect = new StringBuilder(vect0);
    }
    out.close();
}
private static int getResults1() throws FileNotFoundException {
    Scanner scanner = new Scanner(new File(TESTING_VECTORS));
    ArrayList<String> testingValues = new ArrayList<>();
    while(scanner.hasNextLine()){
        testingValues.add(scanner.nextLine());
    }
    scanner.close();
    PrintWriter out = new PrintWriter(SYSTEM_MODEL_RESULTS);
    for (String testingValue : testingValues) {
        out.println(Wrapper1.calculate(testingValue));
    }
    out.close();
    return testingValues.size();
}
private static int getResults2(int capacity) throws FileNotFoundException {
    Scanner scanner = new Scanner(new File(TESTING_VECTORS));
    ArrayList<String> testingValues = new ArrayList<>();
    while(scanner.hasNextLine()){
        testingValues.add(scanner.nextLine());
    }
    scanner.close();
    PrintWriter out = new PrintWriter(MY_SYSTEM_MODEL_RESULTS);
    for (String testingValue : testingValues) {
        out.println(Wrapper2.calculate(testingValue, capacity));
    }
    out.close();
    return testingValues.size();
}
}
```

## 2 RTL-МОДЕЛЬ УСТРОЙСТВА

На данном этапе устройство описывается на уровне соединений между регистрами. Основными вопросами, подлежащими разработке на данном этапе являются: структурная схема RTL-модели, описание модулей RTL-модели, верификация RTL-модели.

### 2.1 Структурная схема RTL-модели

Структурная RTL-схема представлена на Рисунке 2.1.

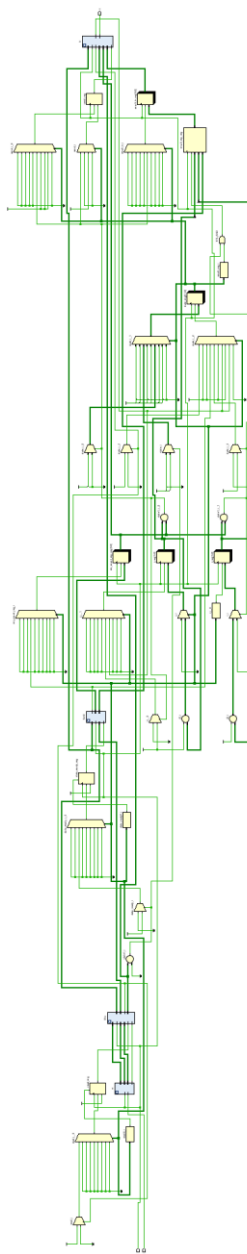


Рисунок 2.1 — RTL-схема

## 2.2 Описание модулей RTL-модели

Реализация вычислительного устройства для минимизации логической функции методом Куайна-Мак-Класки на Verilog состоит из следующих модулей:

1. TOP\_MODULE — модуль верхнего уровня;
8. Fifo — реализация очереди для хранения пришедших значений;
9. fsm — реализация метода Куайна-Мак-Класки в виде автомата;
10. Transmitter — модуль обертка для отправки результата по UART;
11. UART\_TRANSMITTER — модуль для отправки результатов;
12. ASCII\_CODER — преобразователь результата в ascii код;
13. Reciever — модуль обертка для получения входных значений;
14. UART\_RECIEVER — модуль для обработки полученных значений;
15. ASCII\_DECODER — дешифратор ascii кода;
16. REVERSE\_ASCII\_CODER — дополнительный преобразователь ascii кода для тестирования;
17. REVERSE\_ASCII\_DECODER — дополнительный дешифратор ascii кода для тестирования;
18. test — тестовый модуль для сравнения значений полученных на симуляции со значениями системной модели;
19. top\_module\_test — тестовый модуль для проверки корректности принятия и отправки данных по UART;
20. test\_one\_value — тестовый модуль для проверки одного вектора;

FIFO (англ. First In, First Out) — это простая реализация очереди, хранящая пришедшие вектора, количество переменных функции и ошибки, определенные при обработке полученных значений. Из-за особенностей передачи данных по протоколу UART достаточно хранить только 4 последних значения.

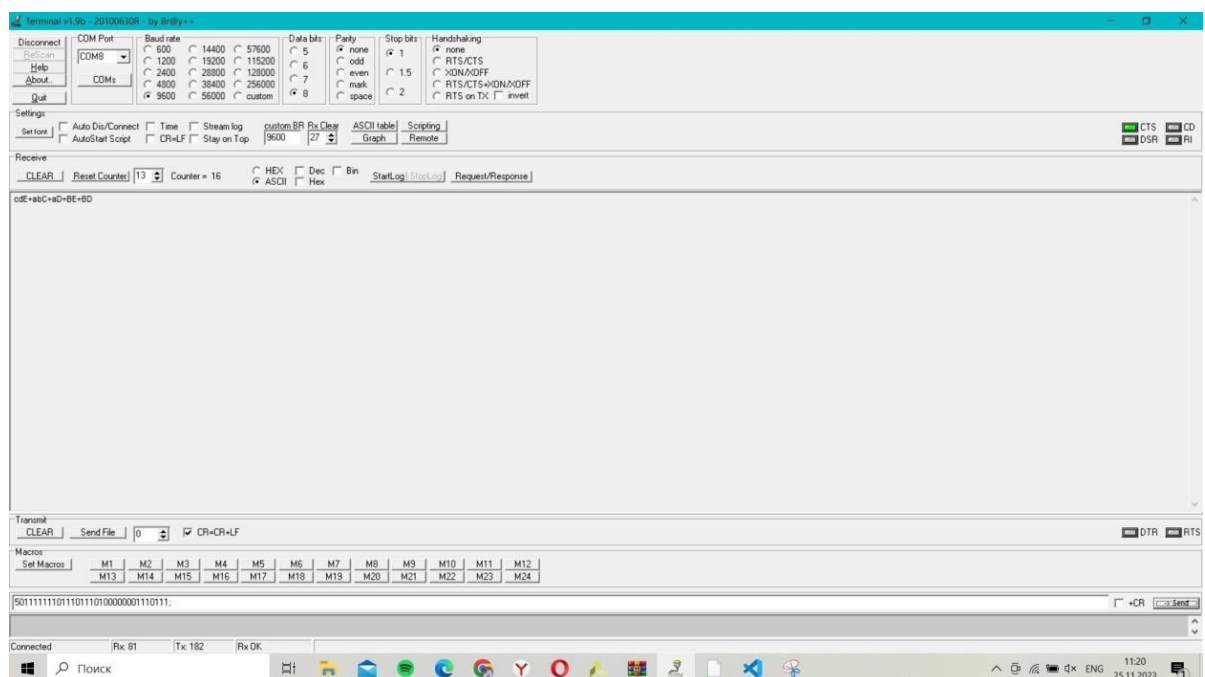


FSM (англ. Finite State Machine) — реализация метода Куайна-МакКласки в виде автомата. Для переноса системной модели на ПЛИС необходимо было разделить вычисления на состояния. Полученный автомат имеет следующие состояния:

1. WAIT\_FOR\_DATA — состояние ожидания новых данных;
2. READ\_IMPLICANTS — запись всех импликант;
3. FILLING\_THE\_INITIAL\_GROUP — заполнение таблицы импликант, разделенных на группы по количеству единиц, для дальнейшего склеивания;
4. COMPARE\_FOR\_MERGING — сравнение двух импликант для склеивания;
5. MERGING — склеивание двух импликант и запись нового значения в таблицу следующего уровня;
6. GET\_COUNT\_OF\_1\_IN\_LOCAL — вычисление количества единиц в импликанте;
7. SET\_MERGED\_VAL\_IN\_NEXT\_COLUMN — запись полученного значения в таблицу следующего уровня;
8. COMPARE\_IMPLICANTS — сравнение двух импликант перед записью нового значения, чтобы избежать повторяющихся значений;
9. ITERATTION\_J — инкремент регистра j;
10. ITERATTION\_I — инкремент регистра i;
11. ITERATTION\_CMP — инкремент регистра cmp\_u;
12. FIND\_SIMPLE\_IMPLICANTS — поиск простых импликант;
13. FILL\_1\_POINTS — заполнение таблицы импликант;
14. COMPARE\_FOR\_Q\_TABLE — проверка значения для внесения в таблицу Куайна;
15. FILL\_QUINE\_TABLE — заполнение таблицы Куайна;
16. CORE\_IMPLICANT\_CHECK — проверка является ли импликанта ядерной;
17. FIND\_CORE\_IMPLICANTS — поиск ядерных импликант;

18. `INDICATE_COVERED_LINES` — поиск импликант, покрытых ядерными;
19. `CREATE_PETRICK_TABLE` — заполнение таблицы Петрика;
20. `PETRICK_METHODS_CALCULATIONS` — применение вариации метода Петрика;
21. `WRITE_RESULT` — запись результата;
22. `READY_RESULT_FLAG` — установка флага готовности результата;
23. `SEND_RESULT` — передача результата в модуль отправки;
24. `RESET` — сброс регистров;

`Transmitter` — модуль обертка для модуля отправки данных по UART. Данный модуль нужен, поскольку результат минимизации функции состоит из нескольких конъюнктов (Рисунок 2.2).



**Рисунок 2.2 — Демонстрация результата работы**

Модуль `Transmitter` отслеживает, какую часть результата необходимо отправить в данный момент, а также отправляет знаки «+» между конъюнктами. К данному модулю подключен модуль `UART_TRANSMITTER`, который отправляет только один символ, данному модулю `Transmitter` по

очереди отправляет всех значения. К UART\_TRANSMITTER подключен ASCII\_CODER для преобразования информации в ascii код.

Модуль Reciver — обертка модуля для получения данных по UART. Данный модуль нужен для принятия значений, отслеживания в них ошибок и записи в модуль Fifo. К модулю подключены UART\_RECIEVER и ASCII\_DECODER. Из UART\_RECIEVER передается каждый полученный символ, а ASCII\_DECODER дешифрирует каждый полученный пакет данных.

REVERSE\_ASCII\_CODER, REVERSE\_ASCII\_DECODER — данные модули используются только для тестирования отправки и чтения значений по UART. REVERSE\_ASCII\_CODER позволяет закодировать в ascii тестовые значения, REVERSE\_ASCII\_DECODER переводит полученный результат из ascii в результат в виде минимизированной логической функции.

## 2.3 Верификация RTL-модели

Проведем верификацию модуля fsm, используя тестовый модуль test\_one\_value.

На Рисунке 2.3 представлена временная диаграмма.

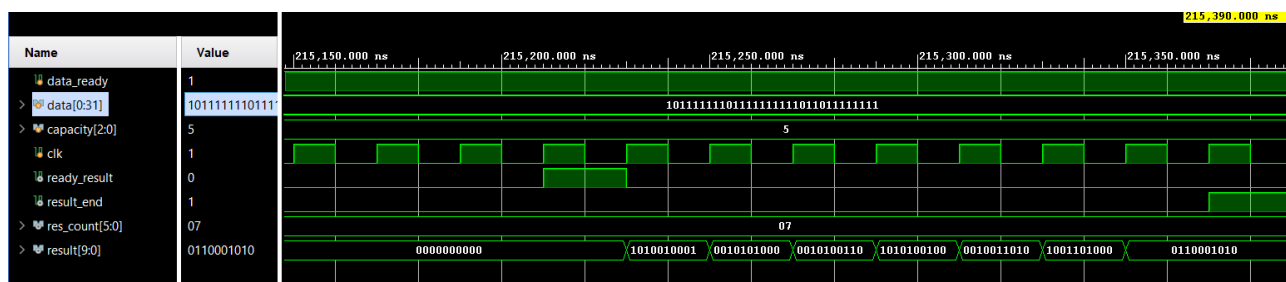


Рисунок 2.3 — Временная диаграмма

На данной временной диаграмме в регистре data представлен тестируемый вектор. Поскольку в тестировании участвует только модуль fsm, на диаграмме продемонстрировано, каким образом fsm отправляет данные модулю Reciever. Когда результат вычислен, отправляется сигнал result\_end, сообщающий о том, что данные будут отправлены начиная со следующего такта. К этому моменту в регистре res\_count уже находится количество конъюнктов в результате. По шине result за один такт передается один

конъюнкт результата. Каждый литерал конъюнкта закодирован 2 битами, поскольку его значением может быть литерал, его инверсия или отсутствие переменной. Максимальная разрядность вектора 5, поэтому размер шины 10 бит. Полученные значения являются верным результатом, в чем можно убедиться при сравнении с данными вектора в системной модели (Рисунок 2.4).

```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" ...
[1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1]
[3, 3, 1, 0, 1, 4] [0, 3, 3, 3, 0, 4] [0, 3, 3, 1, 3, 4] [3, 3, 3, 1, 0, 4] [0, 3, 1, 3, 3, 4] [1, 3, 0, 3, 3, 4] [3, 1, 1, 3, 3, 4]
[CdE, ae, aD, De, aC, Ac, BC]
```

**Рисунок 2.4 — Результат системной модели**

Значения второй строки полностью совпадают с вычисленными в модуле fsm, как уже было сказано одно значение кодируется двумя битами, а последнее значение в системной модели является дополнительным флагом, поэтому незначимо.

В Листинге 2.1 представлен код данного тестового модуля.

*Листинг 2.1 — Тестовый модуль для fsm*

```
module test_one_value();
    reg          data_ready;
    reg [0:31] data;
    reg [2:0] capacity;
    reg          clk;
    wire         ready_result;
    wire         result_end;
    wire [5:0] res_count;
    wire [9:0] result;
    initial begin
        clk = 0;
        data_ready = 1;
        data = {32'b1011111101111111111011011111111};
        capacity = 5;
    end
    always #10 begin
        clk <= ~clk;
    end
    always@(posedge clk) begin
        if(result_end)
            $finish;
    end
    fsm3 m (.data_ready(data_ready), .data(data), .capacity(capacity), .clk(clk),
        .ready_result(ready_result), .result_end(result_end),
        .res_count(res_count), .result(result));
endmodule
```

Рассмотрим верификацию модуля верхнего уровня, используя тестовый модуль top\_module\_test.

На Рисунке 2.5 представлена временная диаграмма.



Рисунок 2.5 — Первая часть диаграммы

На данной диаграмме продемонстрирован прием первого вектора и отправка результата. Регистр value отражает расшифрованный результат (Рисунок 2.6).

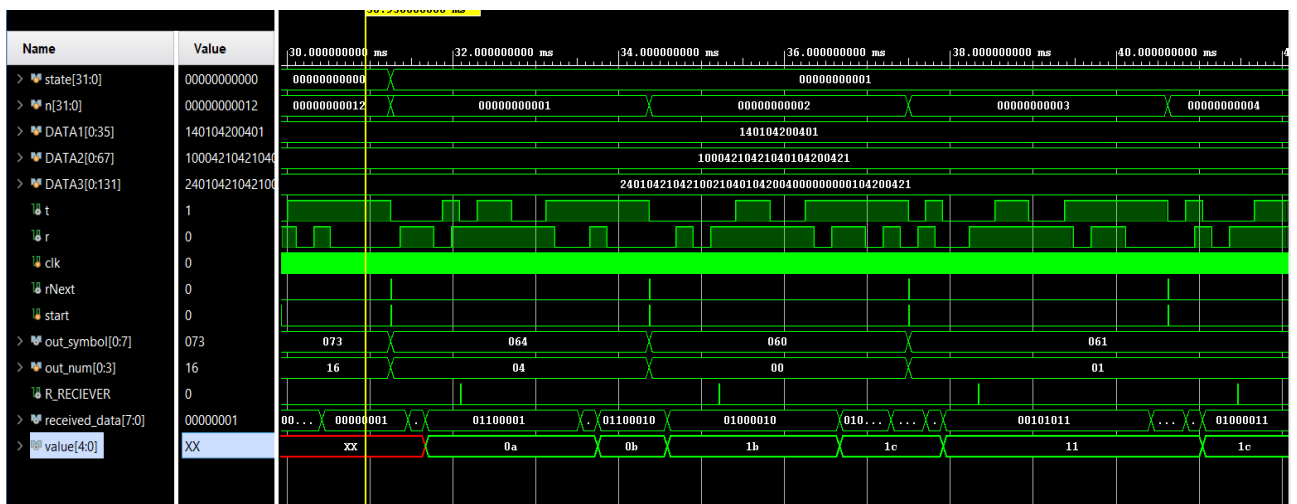


Рисунок 2.6 — Вторая часть диаграммы

Значение считается правильным в момент, когда на R\_RECIEVER приходит 1. Ноль, либо единица в начале обозначают положительное или отрицательное число соответственно. Две единицы — «+». Таким образом, мы видим, что значение получено и результат вычислен верно.

На Рисунке 2.7 представлено отображение ошибок при вводе некорректных данных.

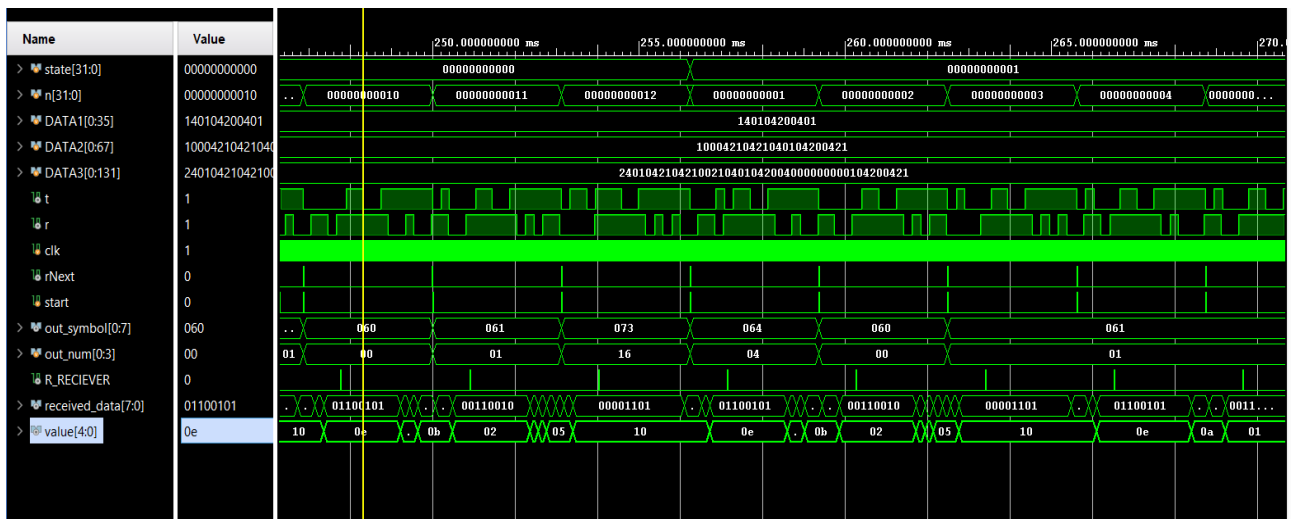


Рисунок 2.7 — Третья часть диаграммы

Код тестового модуля представлен в приложение В.

## 3 РАЗМЕЩЕНИЕ НА ПЛИС

### 3.1 Основные подходы к размещению

Размещение на ПЛИС (программируемых логических интегральных схемах) включает в себя несколько ключевых этапов:

1. Синтез аппаратной схемы: Этот этап включает в себя создание логической схемы, которая определяет функциональность и поведение ПЛИС. В отличие от традиционных микросхем, которые имеют фиксированную функциональность, ПЛИС можно программировать для выполнения различных задач.
2. Размещение и маршрутизация: После синтеза аппаратной схемы необходимо разместить логические элементы на ПЛИС и настроить соединения между ними. Этот процесс называется размещением и маршрутизацией. Размещение определяет физическое расположение логических элементов на ПЛИС, а маршрутизация устанавливает соединения между этими элементами.
3. Генерация конфигурационного файла: После размещения и маршрутизации необходимо сгенерировать конфигурационный файл, который содержит информацию о том, как настроить ПЛИС для работы с созданной аппаратной схемой. Конфигурационный файл может быть сгенерирован с использованием специальных инструментов, предоставляемых производителем ПЛИС.
4. Загрузка конфигурационного файла на ПЛИС: Последний этап программирования ПЛИС – загрузка конфигурационного файла на ПЛИС. Это может быть выполнено с помощью специального программатора, который подключается к ПЛИС и передает конфигурационный файл. После загрузки конфигурационного файла ПЛИС будет настроена для работы с созданной аппаратной схемой.

В Листинге 3.1 представлены все регистры, используемые в модуле fsm.

*Листинг 3.1 — Объявление регистров и цепей модуля fsm*

```
reg [1:0] implicants [0:31][0:5]; //импликаны, заданные вектором
reg [5:0] ci; //count of implicants
reg [5:0] cil2;

reg [1:0] groups0 [0:5][0:9][0:5];
reg [1:0] groups1 [0:4][0:29][0:5];
reg [1:0] groups2 [0:3][0:29][0:5];
reg [1:0] groups3 [0:2][0:19][0:5];
reg [1:0] groups4 [0:1][0:4][0:5];
reg [1:0] groups5 [0:1][0:1][0:5];
//
reg [4:0] cn [0:5][0:5];
reg [2:0] cntr2;
//reg [7:0] z30, k30, t30;
reg [4:0] i, j, z1;
reg [4:0] t, p, z11;
reg [5:0] l2, z12;
reg [2:0] z5, x, k, z14, z7, z3, z2;

reg cf, //comparing flag used to show that it was comparing on this comparing
level
    wf; //write flag
reg [2:0] cmp_out; //result of compare two numbers
reg [2:0] cmp_u; //compare up
reg [2:0] ml; //merging level
reg [2:0] ml_copy1; //merging level
reg [2:0] ml_copy2; //merging level
reg [2:0] ml_copy3; //merging level
reg [1:0] local [0:5]; //new number got by merging
reg [2:0] local_count, t_pos;
wire [2:0] cmp_d; //compare down
assign cmp_d = cmp_u + 1;

reg [2:0] st7_count;
//STATE : SET_MERGED_VAL_IN_NEXT_COLUMN
reg [2:0] _c;

//STATE : FIND_SIMPLE_IMPLICANTS
reg quine_table [0:25][0:25]; //Quine table
reg [1:0] pi [0:31][0:5]; //prime implicants
reg [4:0] cpi; //count of prime implicants
reg [2:0] i11;
reg [2:0] j11;

//STATE : FILL_1_POINTS
reg [1:0] p1 [0:31][0:5];
reg [5:0] cp1;

//STATE : FILL_QUINE_TABLE
reg [5:0] i14, j14;
reg cmp_flag14;

//STATE: FIND_CORE_IMPLICANTS
reg [5:0] i16, z16;
reg [4:0] pos;
reg [5:0] c16;

//STATE: INDICATE_COVERED_LINES
reg [5:0] i17, z17;
```



### Продолжение Листинга 3.1

```
//STATE: CREATE_PETRICK_TABLE
reg [4:0] clpt [0:32];
//
reg petrick_table0 [0:20][0:20];
reg petrick_table1 [0:16][0:20];
reg petrick_table2 [0:12][0:20];
reg petrick_table3 [0:10][0:20];
reg petrick_table4 [0:8][0:20];
reg petrick_table5 [0:6][0:20];
reg petrick_table6 [0:4][0:20];
reg petrick_table7 [0:2][0:20];
reg petrick_table8 [0:1][0:20];
//
reg [4:0] pl; //petrick table level
reg [5:0] z18, l18;
//STATE: PETRICK_METHODS_CALCULATIONS
reg [4:0] l19, t19;
reg [4:0] i19, z19;
reg [4:0] pei; //position of extra implicant
reg [5:0] clc; //count of 1 in column
reg [5:0] max_clc;
//STATE: WRITE_RESULT
reg [1:0] result_reg [0:15][0:4];
reg [4:0] z20;
//STATE: SEND_RESULT
reg [5:0] i22;
```

Результаты размещения согласно основным метрикам представлены на Рисунке 3.1.

#### Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0,023 ns	Worst Hold Slack (WHS): 0,085 ns	Worst Pulse Width Slack (WPWS): 3,750 ns
Total Negative Slack (TNS): 0,000 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 14436	Total Number of Endpoints: 14436	Total Number of Endpoints: 8556
All user specified timing constraints are met.		

Рисунок 3.1 — Показатели slack

## 3.2 Набор ограничений

В Листинге 3.2 представлен набор проектных ограничений для данного проекта.

*Листинг 3.2 — Набор проектных ограничений для данного проекта*

```
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add  
[get_ports clk]  
set_property IOSTANDARD LVCMOS33 [get_ports clk]  
set_property PACKAGE_PIN E3 [get_ports clk]  
  
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN C4} [get_ports r]  
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN D4} [get_ports t]
```

Были подключены тактовый сигнал, трансмиттер и ресивер.

### 3.3 Вариант размещения

На Рисунке 3.2 представлен вариант размещения на ПЛИС, сгенерированный Vivado.

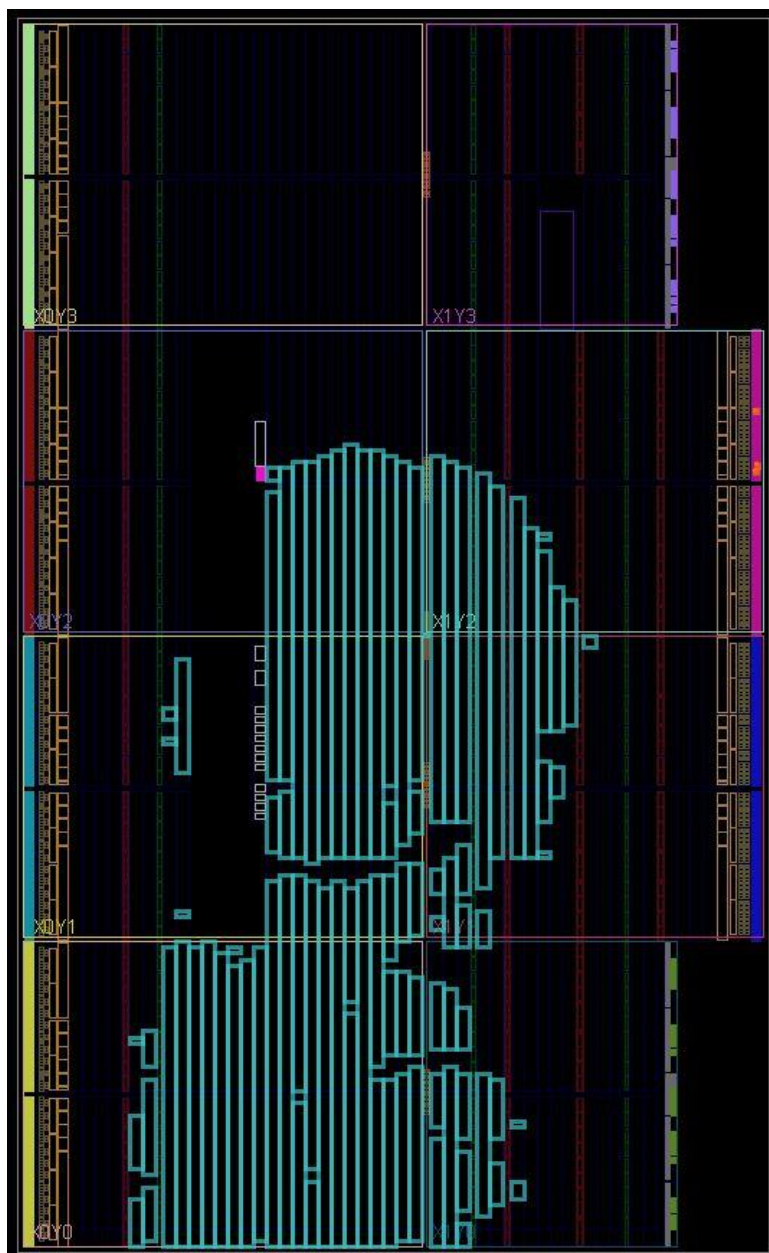


Рисунок 3.2 — Вариант размещения на ПЛИС

Потребляемая мощность представлена на Рисунке 3.3.

#### Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

**Total On-Chip Power:** 0.227 W  
**Design Power Budget:** Not Specified  
**Power Budget Margin:** N/A  
**Junction Temperature:** 26,0°C  
Thermal Margin: 59,0°C (12,8 W)  
Effective  $\theta_{JA}$ : 4,6°C/W  
Power supplied to off-chip devices: 0 W  
Confidence level: Medium  
[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

#### On-Chip Power

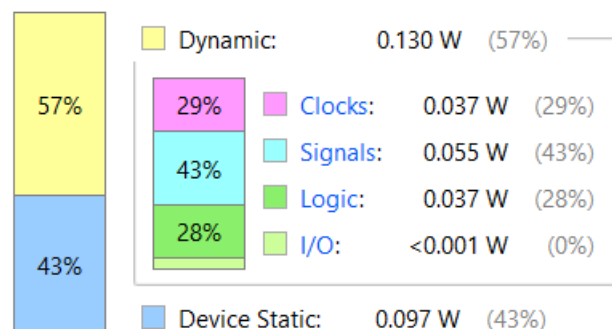


Рисунок 3.3 — Показатели потребляемой мощности

## 4 ТЕСТИРОВАНИЕ

Для тестирования значений, полученных в ходе тестирования, создан модуль на SystemVerilog — test. Код представлен в Листинге 4.1.

*Листинг 4.1 — Модуль test*

```
module test();
reg      data_ready;
reg [0:31] data;
reg [2:0] capacity;
reg      clk;
wire      ready_result;
wire      result_end;
wire [5:0] res_count;
wire [9:0] result;
integer cnt;
reg write;
reg [5:0] c;
integer file;

parameter count_res = 10000;

initial begin
    file = $fopen("results.json","w");
    $fwrite(file, "[");
    write = 0;
    clk = 0;
    data_ready = 1;
    data = {32'b1101000100001001000000000000000000};
    capacity = 4;
    cnt = 0;
    c = 0;
end

always #10 begin
    clk <= ~clk;
end

always@(posedge clk) begin
    if(ready_result) begin
        write <= 1;
        $fwrite(file, "{");
        $fwrite(file, "\"capacity\": %d,", capacity);
        $fwrite(file, "\"func\": \"%b\",", data);
        $fwrite(file, "\"result\" : [");
        c = 0;
        if (data_ready && cnt < count_res + 1) begin
            data_ready <= 0;
        end
    end
    if (write && !result_end) begin
        $write("\"%b\"", result);
        $fwrite(file, "\"%b\"", result);
        c = c + 1;
        if (c < res_count) begin
            $fwrite(file, ",");
        end
    end
    if (result_end) begin
        $fwrite(file, "]");
    end
end
```

#### Продолжение Листинга 4.1

```
data = $urandom();
capacity = $urandom_range(3,5);
case(capacity)
  3 : begin
    data[8:31] <= 24'b000000000000000000000000;
  end
  4 : begin
    data[16:31] <= 16'b0000000000000000;
  end
endcase
data_ready <= 1;
cnt <= cnt + 1;
write <= 0;
$write("\n");
$fwrite(file, "]}");
//$fwrite(file, "%d\n", capacity);
if(cnt < count_res) begin
  $fwrite(file, ",\n");
end
end

if (cnt > count_res) begin
  $fwrite(file, "]}");
  $fclose(file);
  $finish;
end
end

fsm3 m (.data_ready(data_ready), .data(data), .capacity(capacity), .clk(clk),
.res_count(res_count), .result(result));

endmodule
```

В данном модуле генерируются 10000 тысяч случайных векторов для проверки. С помощью функций \$urandom() и \$urandom\_range() генерируются случайный вектор и его кратность.

Далее полученный результат минимизации и исходный вектор записываются в файл в формате json с помощью функций \$write(), \$fwrite().

С помощью программы на языке Java происходит чтение данного файла. Затем для всех векторов вычисляется результат минимизации с помощью системной модели. Полученные значения сравниваются. Тест считается успешным, если все значения совпадают. В Листинге 4.2 представлен код чтения json файла.

#### Листинг 4.2 — Чтение json файла

```
package test_verilog_results;

import my_system_model.Wrapper2;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import org.junit.jupiter.api.Test;

import java.io.FileReader;
import java.io.IOException;

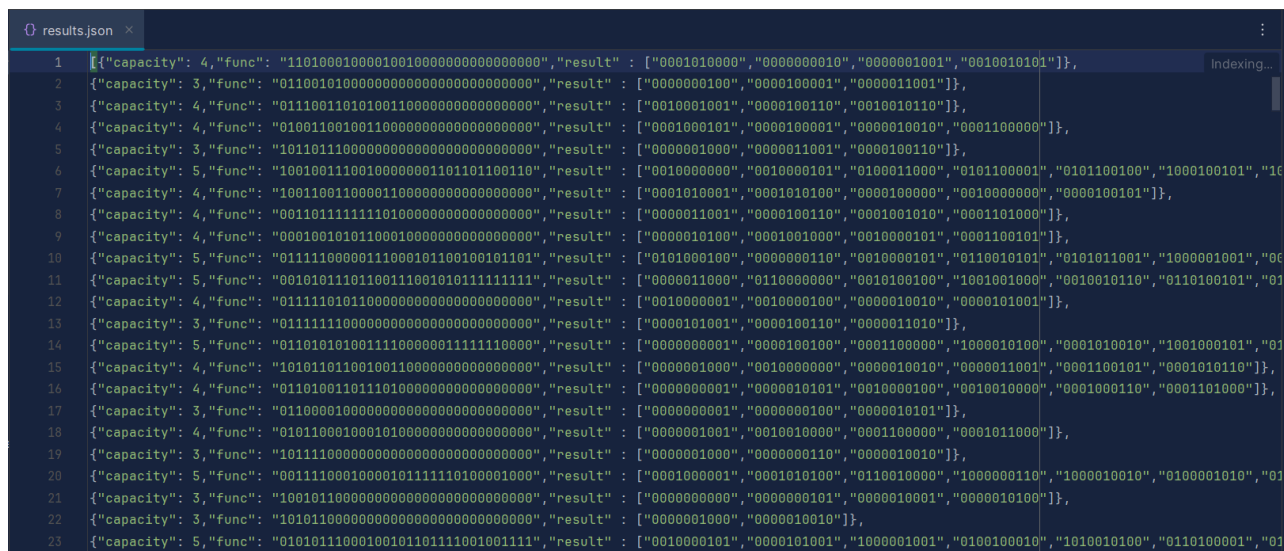
import static org.junit.jupiter.api.Assertions.assertEquals;

public class TestVerilogResults {
    private static final String PATH =
"C:\\\\Users\\paras\\\\Xilinx\\\\coursework2\\\\coursework2.sim\\\\sim_1\\\\behav\\\\xsim\\\\re
sults.json";
    private Result[] results;

    public void readVerilog(){
        Object o = null;
        try {
            o = new JSONParser().parse(new FileReader(PATH));
            JSONArray j = (JSONArray) o;
            JSONObject object;
            JSONArray arr_res;
            String[] arr_res_str;
            Result[] results = new Result[j.size()];
            for(int i = 0; i < j.size(); i++){
                object = (JSONObject) j.get(i);
                arr_res = (JSONArray) object.get("result");
                arr_res_str = new String[arr_res.size()];
                for(int k = 0; k < arr_res.size(); k++){
                    arr_res_str[k] = (String) arr_res.get(k);
                }
                results[i] = new Result((long) object.get("capacity"),
                    (String) object.get("func"),
                    arr_res_str);
            }
            this.results = results;
        } catch (IOException | ParseException e) {
            e.printStackTrace();
        }
    }

    @Test
    public void testCheckMinimization(){
        readVerilog();
        Result res;
        for (Result result : results) {
            res = result;
            String sys_model_res = Wrapper2.calculate(res.func, (int)
res.capacity);
            System.out.println(sys_model_res + " " + res.resultAsString + " " +
res.func + " " + res.capacity);
            if(sys_model_res.equals("0"))
                assertEquals("", res.resultAsString);
            else
                assertEquals(sys_model_res, res.resultAsString);
        }
    }
}
```

Пример полученного json файла представлен на Рисунке 4.1.



```
1 [{"capacity": 4, "func": "11010001000010010000000000000000", "result": ["0001010000", "0000000010", "0000001001", "0010010101"]},
2 {"capacity": 3, "func": "01100101000000000000000000000000", "result": ["000000100", "0000100001", "0000011001"]},
3 {"capacity": 4, "func": "01110011010100110000000000000000", "result": ["0010001001", "0000100110", "0010010110"]},
4 {"capacity": 4, "func": "01001100100110000000000000000000", "result": ["0001000101", "0000100001", "0000010010", "0001100000"]},
5 {"capacity": 3, "func": "10110111000000000000000000000000", "result": ["0000001000", "0000011001", "0000100110"]},
6 {"capacity": 5, "func": "10010011100100000001101101100110", "result": ["0010000000", "0010000101", "0100011000", "0101100001", "0101100100", "1000100101", "1000100101", "1000100101"]},
7 {"capacity": 4, "func": "10011001100001100000000000000000", "result": ["0001010001", "0001010100", "0000100000", "0010000000", "0000100101"]},
8 {"capacity": 4, "func": "00110111111110100000000000000000", "result": ["0000011001", "0000100110", "0001001010", "0001101000"]},
9 {"capacity": 4, "func": "00010010101100010000000000000000", "result": ["0000010100", "0001001000", "0010000101", "0001100101"]},
10 {"capacity": 5, "func": "01111100000111000101100100101101", "result": ["0101000100", "0000000110", "0010000101", "0110010101", "0101011001", "1000001001", "0000000101", "0000000101", "0000000101", "0000000101"]},
11 {"capacity": 5, "func": "00101011101100111001010111111111", "result": ["0000001000", "0110000000", "0010100100", "1001001000", "0010010110", "0110100101", "0110100101", "0110100101", "0110100101", "0110100101"]},
12 {"capacity": 4, "func": "01111101011000000000000000000000", "result": ["0010000001", "0010000100", "0000010010", "0000101001"]},
13 {"capacity": 3, "func": "01111110000000000000000000000000", "result": ["0000101001", "0000100110", "0000011010"]},
14 {"capacity": 5, "func": "01101010100111100000011111110000", "result": ["0000000001", "0000100100", "0001100000", "1000010100", "0001010010", "1001000101", "0001000101", "0001000101", "0001000101", "0001000101"]},
15 {"capacity": 4, "func": "10101101100100110000000000000000", "result": ["0000001000", "0010000000", "0000010010", "0000011001", "0001100101", "0001010110"]},
16 {"capacity": 4, "func": "01101001101110100000000000000000", "result": ["0000000001", "0000010101", "0010000100", "0010010000", "0001000110", "0001101000"]},
17 {"capacity": 3, "func": "01100001000000000000000000000000", "result": ["0000000001", "0000000100", "0000010101"]},
18 {"capacity": 4, "func": "01011000100010100000000000000000", "result": ["0000001001", "0010010000", "0001100000", "0001011000"]},
19 {"capacity": 3, "func": "10111100000000000000000000000000", "result": ["0000001000", "0000000110", "0000010010"]},
20 {"capacity": 5, "func": "00111100010000101111110100001000", "result": ["0001000001", "0001010100", "0110010000", "1000000110", "1000010010", "0100001010", "0100001010", "0100001010", "0100001010", "0100001010"]},
21 {"capacity": 3, "func": "10010110000000000000000000000000", "result": ["0000000000", "0000000101", "0000010001", "0000010100"]},
22 {"capacity": 3, "func": "10101100000000000000000000000000", "result": ["0000001000", "0000010010"]},
23 {"capacity": 5, "func": "01010111000100101101111001001111", "result": ["0010000101", "0000101001", "1000001001", "0100100010", "1010010100", "0110100001", "0110100001", "0110100001", "0110100001", "0110100001"]}]
```

Рисунок 4.1 — Пример json файла

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы было спроектировано вычислительное устройство для минимизации логической функции методом Куайна-Мак-Класски.

Был проведен анализ предметной области. Разработан набор модулей, описывающих процесс минимизации функции методом Куайна-Мак-Класски. Разработан набор модулей, описывающих процессы приёма и обработки данных по протоколу UART. Разработан модуль верхнего уровня, реализующий управление для совместной работы всех модулей. Проведена верификация. Составлена отчетная документация по проделанной работе.

Было продемонстрировано, что ПЛИС, благодаря своей гибкости и программируемости, позволяют создавать сложные цифровые схемы с высокой производительностью. Они обладают большим количеством входных и выходных портов, что позволяет создавать устройства с большим количеством входов и выходов.

Метод Куайна-Мак-Класски является эффективным инструментом для минимизации булевых выражений, что может быть полезно при проектировании цифровых схем и микропроцессоров. Этот метод позволяет упростить логические функции и уменьшить количество используемых логических элементов, что приводит к увеличению скорости работы устройства.

Однако, стоит отметить, что хотя метод Куайна-Мак-Класски является эффективным инструментом для минимизации булевых выражений, он не всегда может быть применен для всех типов логических функций. В некоторых случаях, другие методы оптимизации могут быть более подходящими.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Тарасов И.Е. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. — М.: Горячая линия — Телеком, 2021. — 538 с.: ил.
2. Антик М.И. Математическая логика и программирование в логике [Электронный ресурс]: Учебное пособие / Антик М.И., Бражникова Е.В.— М.: МИРЭА – Российский технологический университет, 2018. — 1 электрон. опт. диск (CD-ROM).
3. Смирнов С.С. Информатика [Электронный ресурс]: Методические указания по выполнению практических и лабораторных работ / С.С. Смирнов — М., МИРЭА — Российский технологический университет, 2018. — 1 электрон. опт. диск (CD-ROM).
4. Паттерсон Д., Хеннесси Дж. Архитектура компьютера и проектирование компьютерных систем. 4-е изд. СПб.: Питер, 2012. – ISBN 978-5-459-00291-1.
5. Харрис Дэвид М., Харрис Сара Л. Цифровая схемотехника и архитектура компьютера. Издательство: ДМК-Пресс, 2018 г.
6. Максфилд К. Проектирование на ПЛИС. Курс молодого бойца. – М.: Издательский дом «Додэка-XXI», 2007. – 408 с.: илл. (Серия «Программируемые системы»).
7. Макконнелл С. Совершенный код. Мастер-класс / Пер. с англ. – М.: Издательство «Русская редакция», 2013. – 896 с.: ил.
8. Шафер Д., Фатрелл Р., Шафер Л. Управление программными проектами: достижение оптимального качества при минимуме затрат: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 1136 с.: ил. – Парал. тит. англ.
9. Таненбаум Эндрю, Остин Т. Архитектура компьютера. Издательство: Питер, 2019 г. Серия: Классика computer science. ISBN: 978-5-4461-1103-9, 816 с.

10. Вонг Б.П., Миттал А., Цао Ю., Старр Г. Нано-КМОП-схемы и проектирование на физическом уровне. Москва: Техносфера, 2014. – 432 с., ISBN 978-5-94836-377-6.
11. Джонс М.Х. Электроника – практический курс. Изд. 3-е, исправленное. Москва: Техносфера, 2021. – 512 с. ISBN 978-5-94836-341-7.
12. Белоус А.И., Красников Г.Я., Солодуха В.А. Основы проектирования субмикронных микросхем. Москва: Техносфера, 2020. – 782 с. ISBN 978-5-94836-603-6.
13. Рабан, Жан.М., Чандракасан, А., Николич, Б. Цифровые интегральные схемы. Методология проектирования. 2-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2016. – 912 с.: ил. – Парал. тит. англ. ISBN 978-5-8459-1116-2 (рус.)
14. Савельев А.Я. Основы информатики. — Москва: Издательство МГТУ им. Н.Э. Баумана, 2001. — С. 232—239. — 328 с. — (Информатика в техническом университете). — ISBN 5703815150.
15. Методические указания по ПР № 2 — URL: [onlineedu.mirea.ru/mod/resource/view.php?id=409130](https://onlineedu.mirea.ru/mod/resource/view.php?id=409130) (Дата обращения: 23.09.2022).
16. Реализация метода Куайна-Мак-Класки на Java — URL: <https://github.com/ironmaurus/mcquine> (Дата обращения: 13.10.2023).

## **ПРИЛОЖЕНИЯ**

Приложение А — Системная модель

Приложение Б — Реализация на Verilog

Приложение В — Модуль top\_module\_test

# Приложение А.1

## Системная модель

### Листинг А.1 — Системная модель

```
Системная модель
package my_system_model;

import java.util.ArrayList;
import java.util.Arrays;

public class Run {
    //Ввод данных с платы:
    static byte n0 = 5; //разрядность функции
    static byte[] func =
{0,1,1,0,0,1,1,1,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0}; //вектор

    static byte n1 = 5; //разрядность функции
    static byte[] func1 =
{1,1,1,0,0,1,1,1,1,1,0,1,1,1,0,0,0,1,0,0,1,1,0,0,0,1,0,0,1,1,0,0}; //вектор

    static byte n2 = 5; //разрядность функции
    static byte[] func2 =
string_to_array("00101000101100010010100010110001"); //вектор

    static byte n3 = 5; //разрядность функции
    static byte[] func3 =
string_to_array("11111111111111111111111111111111"); //вектор

    static byte n4 = 2; //разрядность функции
    static byte[] func4 = string_to_array("11"); //вектор

    public static void main(String[] args) {
        //quineMcCluskey(func, n);
        ArrayList<byte[]> quineMcCluskey = quineMcCluskey(func3, n3);
        System.out.println(Arrays.toString(func3));
        for (byte[] arr : quineMcCluskey) {
            System.out.println(Arrays.toString(arr));
        }
    }
    private static byte[] string_to_array(String str){
        byte[] arr = new byte[str.length()];
        for (int i = 0; i < str.length(); i++){
            arr[i] = (byte) Integer.parseInt(String.valueOf(str.charAt(i)));
        }
        return arr;
    }
    static ArrayList<byte[]> quineMcCluskey(byte[] func, int n){
        byte k = (byte) Math.pow(2, n);
        //Массив регистров со всеми импlicantsами, размер одного регистра - n,
размер массива 2**n
        byte[][] implicants = new byte[k][n];
        byte ci; //Кол-во импlicants
        //Получаем список импlicants
        ci = 0;
        String binary;
        int bs;
        for(int i = 0; i < k; i++){
            if(func[i] == 1) {
                binary = Integer.toBinaryString(i);
                bs = binary.length();
            }
        }
    }
}
```

### Продолжение Листинга A.1

```
        for(int j = n - bs; j < n; j++){
            implicants[ci][j] = (byte)
Integer.parseInt(binary.substring(j - (n - bs), j+1 - (n - bs)));
        }
        ci++;
    }
}

//Группы
byte[][][] groups = new byte[6][k][k][n+1];

//Распределение по группам
//Группа с 5 переменными
int c;
byte[][] cn = new byte[6][n+1];

for(int i = 0; i < ci; i++){
    c = count_of_1(implicants[i], n);
    for(int j = 0; j < n; j++){ //Это ок, потому что системная модель
        пишется с учетом будущей реализации на verilog
        groups[0][c][cn[0][c]][j] = implicants[i][j];
    }
    cn[0][c]++;
}

//Переменные для склеивания
byte cmp_out; //result of compare two numbers
byte[] local; //new number got by merging
byte cmp_d; //compare down
byte cmp_u; //compare up
boolean wf; //write flag
byte ml = 0; //merging level
boolean cf = true; //comparing flag used to show that it was comparing
on this comparing level

//Склеивание подгрупп в цикле

while(cf) {
    cf = false;

    for (cmp_u = 0; cmp_u < n; cmp_u++) {
        cmp_d = (byte) (cmp_u + 1);
        for (int i = 0; i < cn[ml][cmp_u]; i++) {
            for (int j = 0; j < cn[ml][cmp_d]; j++) {
                //Функции сравнения вынести в виде task'ов
                cmp_out = compare_for_merging(groups[ml][cmp_u][i],
groups[ml][cmp_d][j], n); //больше 0 если можно склеить
                if (cmp_out >= 0) {
                    cf = true;
                    local = groups[ml][cmp_u][i].clone();
                    local[cmp_out] = 3;
                    local[n] = 0;

                    groups[ml][cmp_u][i][n] = 4;
                    groups[ml][cmp_d][j][n] = 4;

                    c = count_of_1(local, n);

                    wf = true;
                    for (int p = 0; p < cn[ml + 1][c]; p++) {
```

```

        if (compare_implicants(local, groups[ml +
1][c][p], n)) {
            wf = false;
            break;
        }
    }

    if (wf) {
        groups[ml + 1][c][cn[ml + 1][c]] = local;
        cn[ml + 1][c]++;
    }
}
}

ml++;
}

//output_cmd(groups, cn);

//Таблица Квайна
int[][] quine_table = new int[32][32];

//Массив простых импликант
byte[][] pi = new byte[32][6]; //prime implicants

//Поиск всех простых импликант
int cpi = 0; //count of prime implicants

for (int i = 0; i <= n; i++) {
    for (int j = 0; j <= n; j++) {
        for (int b = 0; b < cn[i][j]; b++){
            if (groups[i][j][b][n] == 0){
                pi[cpi] = groups[i][j][b].clone();
                cpi++;
            }
        }
    }
}

//Массив 1-точек функции
byte[][] p1 = new byte[32][6];

int cpl = 0; //count of point 1

for (int i = 0; i < n+1; i++){
    for (int j = 0; j < cn[0][i]; j++) {
        p1[cpl] = groups[0][i][j].clone();
        p1[cpl][n] = 0;
        cpl++;
    }
}

//Заполнение таблицы Квайна
for(int i = 0; i < ci; i++){
    for(int j = 0; j < cpi; j++){
        if (compare_for_q_table(p1[i], pi[j], n))
            quine_table[i][j] = 1;
    }
}

```

### Продолжение Листинга А.1

```
//Поиск ядерных импликант
int cicr; //core_implicant_check result
//int cci = 0;
for(int i = 0; i < ci; i++){
    cicr = core_implicant_check(quine_table[i], cpi);
    if(cicr >= 0) {
        pi[cicr][n] = 4;
        //cci++;
    }
}

//Отмечаем строки, перекрытые ядерными импликантами
for (int i = 0; i < cpi; i++){
    if(pi[i][n] == 4){
        for (int j = 0; j < cpl; j++){
            if (quine_table[j][i] == 1){
                pl[j][n] = 4;
            }
        }
    }
}

//Создадим отдельную табличку для Петрика из оставшихся простых
импликант
int[] clpt = new int[32];
int[][][] petrick_table = new int[32][32][32];
int l = 0; //petrick table level
for (int i = 0; i < cpl; i++){
    if(pl[i][n] == 0) {
        petrick_table[0][clpt[0]] = quine_table[i].clone();
        clpt[0]++;
    }
}

//Далее будем использовать одну из вариаций метода петрика.
int pei = 0; //position of extra implicant
//int cei = 0; //count of extra implicants
int clc = 0; //count of 1 in column
int max_clc = 0;

if(clpt[0] > 0) {
    while (true) {
        for (int i = 0; i < cpi; i++) {
            for (int j = 0; j < clpt[l]; j++) {
                if (petrick_table[l][j][i] == 1) {
                    clc++;
                }
            }
            if (clc > max_clc) {
                max_clc = clc;
                pei = i;
            }
            clc = 0;
        }

        pi[pei][n] = 4;
        for (int i = 0; i < clpt[l]; i++) {
            if (petrick_table[l][i][pei] != 1) {
                petrick_table[l + 1][clpt[l + 1]] =
petrick_table[l][i].clone();
                clpt[l + 1]++;
            }
        }
        if (clpt[l + 1] == 0) {
```

```
        break;
    }
    l++;
    max_clc = 0;

}

}

ArrayList<byte[]> res = new ArrayList<>();
for (int i = 0; i < cpi; i++){
    if(pi[i][n] == 4) {
        res.add(pi[i]);
    }
}
return res;
}

static int count_of_1(byte[] bin_num, int n){
    int c = 0;
    for (int j = 0; j < n; j++){
        if (bin_num[j] == 1)
            c++;
    }
    return c;
}

static int core_implicant_check(int[] a, int csi){
    int pos = 0;
    int c = 0;
    for (int i = 0; i < csi; i++){
        if(a[i] == 1) {
            c++;
            pos = i;
        }
    }
    if (c == 1)
        return pos;
    else
        return -1;
}

static byte compare_for_merging(byte[] f1, byte[] f2, int n){
    int local_count = 0;
    byte t_pos = 0;
    for(byte t = 0; t < n; t++){
        if (f1[t] != f2[t]){
            local_count++;
            t_pos = t;
        }
    }
    if (local_count == 1)
        return t_pos;
    else
        return -1;
}

static boolean compare_implicants(byte[] f1, byte[] f2, int n){
    for (int i = 0; i < n; i++){
        if (f1[i] != f2[i]){
```



*Продолжение Листинга А.1*

```
        return false;
    }
    return true;
}

static boolean compare_for_q_table(byte[] p1, byte[] si, int n) {
    for (int i = 0; i < n; i++){
        if (!(p1[i] == si[i] || si[i] == 3)){
            return false;
        }
    }
    return true;
}
}
```

## Приложение А.2

### Реализация функций для тестирования системной модели

*Листинг А.2 — Реализация функций для тестирования системной модели*

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;
import static java.lang.Math.pow;
import static org.junit.jupiter.api.Assertions.assertTrue;
public class CheckMinimizedFunction {
    protected static final String BAD_METRIC_LINES =
"src/test/resources/badMetricLines.txt";
    private ArrayList<String> testingValues1;
    private ArrayList<String> testingValues2;
    private ArrayList<String> inputVectors;
    private int capacity;
    private int testValuesCount;

    public ArrayList<String> getTestingValues1() {
        return testingValues1;
    }

    public ArrayList<String> getTestingValues2() {
        return testingValues2;
    }

    public ArrayList<String> getInputVectors() {
        return inputVectors;
    }

    public void launch(int capacity, int testValuesCount){
        this.capacity = capacity;
        this.testValuesCount = testValuesCount;
        try {
            readResultArrays();
            readInputVector();
        } catch (FileNotFoundException e) {
            System.out.println("Can't read results and input vectors files");
            e.printStackTrace();
        }
    }

    public void testCheckMinimization(){
        boolean res;
        for (int i = 0; i < testValuesCount; i++) {
            res = checkOneVector(i, 0);
            System.out.println(res + " :" + i);
            assertTrue(res);
        }
    }

    public void testCheckMinimizationMy(){
        boolean res;
        for (int i = 0; i < testValuesCount; i++) {
            res = checkOneVector(i, 1);
            assertTrue(res);
        }
    }
}
```

```

    }
}

private void readResultArrays() throws FileNotFoundException {
    Scanner scanner = new Scanner(new File(MainTest.SYSTEM_MODEL_RESULTS));
    testingValues1 = new ArrayList<>();
    while(scanner.hasNextLine()){
        testingValues1.add(scanner.nextLine());
    }
    scanner.close();
    scanner = new Scanner(new File(MainTest.MY_SYSTEM_MODEL_RESULTS));
    testingValues2 = new ArrayList<>();
    while(scanner.hasNextLine()){
        testingValues2.add(scanner.nextLine());
    }
    scanner.close();
}

private void readInputVector() throws FileNotFoundException {
    Scanner scanner = new Scanner(new File(MainTest.TESTING_VECTORS));
    inputVectors = new ArrayList<>();
    while(scanner.hasNextLine()){
        inputVectors.add(scanner.nextLine());
    }
    scanner.close();
}

private boolean checkOneVector(int index, int mode){
    boolean result = false;
    boolean final_result = true;

    if (index > inputVectors.size() || index < 0){
        System.out.println("Error: wrong index");
        return false;
    }

    String[] implss = mode == 0 ? testingValues1.get(index).split("\\\\+") :
testingValues2.get(index).split("\\\\+");

    ArrayList<char[]> impls = new ArrayList<>();
    for (String imp : implss){
        impls.add(imp.toCharArray());
    }

    StringBuilder buf;

    for (int i = 0; i < pow(2, capacity); i++){
        result = false;
        buf = new StringBuilder(Integer.toBinaryString(i));
        buf.insert(0, "0000".toCharArray(), 0, capacity - buf.length());
        for (char[] imp : impls){
            result = checkOneImp(imp, buf.toString()) || result;
            //System.out.print(Arrays.toString(imp) + " ");
        }
        System.out.print(i + " " + result + " " +
inputVectors.get(index).charAt(i) + " " + buf.toString()
        + " " + "; ");
        final_result = final_result && ((result ? '1' : '0') ==
inputVectors.get(index).charAt(i));
    }
    System.out.println();
    return final_result;
}

```

```
}

private boolean checkOneImp(char[] imp, String index){
    boolean res = true;
    for (char c : imp){
        switch (c) {
            case ('a') -> res = index.charAt(0) == '0' && res;
            case ('A') -> res = index.charAt(0) == '1' && res;
            case ('b') -> res = index.charAt(1) == '0' && res;
            case ('B') -> res = index.charAt(1) == '1' && res;
            case ('c') -> res = index.charAt(2) == '0' && res;
            case ('C') -> res = index.charAt(2) == '1' && res;
            case ('d') -> res = index.charAt(3) == '0' && res;
            case ('D') -> res = index.charAt(3) == '1' && res;
            case ('e') -> res = index.charAt(4) == '0' && res;
            case ('E') -> res = index.charAt(4) == '1' && res;
            case ('0') -> res = false;
            case ('1') -> res = true;
        }
    }
    return res;
}

public boolean checkMetricOfResultFunction() throws FileNotFoundException {
    PrintWriter out = new PrintWriter(BAD_METRIC_LINES);

    for (int i = 0; i < testValuesCount; i++){
        if(testingValues1.get(i).length() !=
testingValues2.get(i).length()){
            out.println(i+1 + " val1: " + testingValues1.get(i).length() + "
" + testingValues1.get(i) +
                " my val: " + testingValues2.get(i).length() + " " +
testingValues2.get(i));
            out.println(inputVectors.get(i));
            out.close();
            return false;
        }
    }
    out.close();
    return true;
}
}
```

## Приложение Б.1

### Модуль TOP\_MODULE

*Листинг Б.1 — Реализация модуля верхнего уровня*

```
module TOP_MODULE(
    input clk, r,
    output t
);
//Fifo
reg read;
wire write;
wire [0:31] data_in, data_out;
wire [2:0] capacity_in, capacity_out;
wire [2:0] error_reg_in, error_reg_out;
wire empty;

//fsm
reg data_ready;
wire ready_result, result_end;
wire [5:0] res_count;
wire [9:0] result;

//Transmitter
reg send;
wire ready_next;

parameter CHECK_FIFO = 0;
parameter READ_FIFO = 1;
parameter WAIT_FOR_RESULT = 2;
parameter SAVE_RESULT = 3;
parameter SEND_RESULT = 4;
parameter SEND_RESULT_1 = 5;
parameter SEND_ERROR_RESULT = 6;
parameter END_OF_ERROR = 7;
parameter READ_0 = 8;

integer state;
reg [9:0] result_reg [0:15];
reg [5:0] res_count_reg;
reg [5:0] i;
reg [5:0] j;
reg [9:0] result_tr;

initial begin
    state = 0;
    read = 0; //Fifo
    data_ready = 0; // fsm
    send = 0; // Transmitter
    i = 0; j = 0;
    result_tr = 0;
    res_count_reg = 0;
end

always@(posedge clk)begin
    case(state)
        CHECK_FIFO: begin
            if(!empty) begin
                read <= 1;
                state <= READ_0;
            end
        end
    end
end
```

*Продолжение Листинга Б.1*

```
    READ_0: begin
        read <= 0;
        state <= READ_FIFO;
    end
    READ_FIFO: begin
        if (error_reg_out != 0) begin
            state <= SEND_ERROR_RESULT;
        end
        else begin
            data_ready <= 1;
            state <= state + 1;
        end
    end
    WAIT_FOR_RESULT: begin
        data_ready <= 0;
        if (ready_result) begin
            res_count_reg <= res_count;
            state <= state + 1;
        end
    end
    SAVE_RESULT: begin
        if (i < res_count_reg) begin
            result_reg[i] <= result;
            i <= i + 1;
        end
        else begin
            state <= state + 1;
            i <= 0;
        end
    end
    SEND_RESULT: begin
        if (j < res_count_reg) begin
            result_tr <= result_reg[j];
            send <= 1;
            state <= state + 1;
        end
        else begin
            state <= 0;
            j <= 0;
        end
    end
    SEND_RESULT_1: begin
        send <= 0;
        if (ready_next) begin
            j <= j + 1;
            state <= state - 1;
        end
    end
    SEND_ERROR_RESULT: begin
        send <= 1;
        state <= state + 1;
    end
    END_OF_ERROR: begin
        send <= 0;
        if (ready_next) begin
            state <= CHECK_FIFO;
        end
    end
endcase
end
```

*Продолжение Листинга Б.1*

```
Fifo fifo (write, read, clk, data_in, capacity_in, error_reg_in, data_out,  
capacity_out, error_reg_out, empty);  
fsm3 fsm3 (data_ready, data_out, capacity_out, clk, ready_result, result_end,  
res_count, result);  
Transmitter tr (result_tr, res_count_reg, capacity_out, error_reg_out, empty,  
send, clk, t, ready_next);  
Reciever rc (r, clk, data_in, capacity_in, error_reg_in, write);  
endmodule
```

## Приложение Б.2

### Модуль Fifo

*Листинг Б.2 — Реализация модуля Fifo*

```
module Fifo(input write, read, clk,  
            [0:31] data_in, [2:0] capacity_in, [2:0] error_reg_in,  
            output reg [0:31] data_out, reg [2:0] capacity_out, reg [2:0]  
            error_reg_out, reg empty);  
  
    reg [0:31] data_arr      [0:3];  
    reg [2:0]  capacity_arr  [0:3];  
    reg [2:0]  error_reg_arr [0:3];  
  
    reg [1:0] wp;  
    reg [1:0] rp;  
    reg [2:0] c;  
  
    initial begin  
        wp = 0;  
        rp = 0;  
        empty = 1;  
        c = 0;  
    end  
  
    always@(posedge clk) begin  
        if(read && !empty) begin  
            data_out <= data_arr[rp];  
            capacity_out <= capacity_arr[rp];  
            error_reg_out <= error_reg_arr[rp];  
            rp <= rp + 1;  
            c <= c - 1;  
            if (c - 1 == 0)  
                empty <= 1;  
        end  
        if(write) begin  
            data_arr[wp] <= data_in;  
            capacity_arr[wp] <= capacity_in;  
            error_reg_arr[wp] <= error_reg_in;  
            wp <= wp + 1;  
            if (c < 4)  
                c <= c + 1;  
            if (wp + 1 == rp)  
                rp <= rp + 1;  
            empty <= 0;  
        end  
    end  
  
endmodule
```



## Приложение Б.3

### Модуль fsm3

Листинг Б.3 — Реализация модуля fsm3

```
module fsm3 (
    input          data_ready,
    input          [0:31] data,
    input          [2:0] capacity,
    input          clk,

    output reg     ready_result,
    output reg     result_end,
    output reg [5:0] res_count,
    output reg [9:0] result //10 bits = 5 * 2 ; 5 - переменных в импиканте,
                          //2 - бита для кодирования одного символа в
импиканте (0,1,-,*)
);

parameter WAIT_FOR_DATA = 0;
parameter READ_IMPLICANTS = 1;
parameter FILLING_THE_INITIAL_GROUP = 2;
parameter COMPARE_FOR_MERGING = 3;
parameter MERGING = 4;
parameter GET_COUNT_OF_1_IN_LOCAL = 5;
parameter SET_MERGED_VAL_IN_NEXT_COLUMN = 6; //set merged value in next column
of merging table
parameter COMPARE_IMPLICANTS = 7;
parameter ITERATTION_J = 8; //TODO: убрать вторую t
parameter ITERATTION_I = 9;
parameter ITERATTION_CMP = 10;
parameter FIND_SIMPLE_IMPLICANTS = 11;
parameter FILL_1_POINTS = 12;
parameter COMPARE_FOR_Q_TABLE = 13;
parameter FILL_QUINE_TABLE = 14;
parameter CORE_IMPLICANT_CHECK = 15;
parameter FIND_CORE_IMPLICANTS = 16;
parameter INDICATE_COVERED_LINES = 17;
parameter CREATE_PETRICK_TABLE = 18;
parameter PETRICK_METHODS_CALCULATIONS = 19;
parameter WRITE_RESULT = 20;
parameter READY_RESULT_FLAG = 21;
parameter SEND_RESULT = 22;
parameter RESET = 31;

reg [4:0] state;
initial begin
    state = RESET;
end
`include "fsm_regs.vh"

always@(posedge clk) begin
    case(state)
        WAIT_FOR_DATA: begin
            if(data_ready) begin
                state <= state + 1;//READ_IMPLICANTS
            end
        end
        READ_IMPLICANTS: begin
            if(z1 == 31) begin
                state <= state + 1;//FILLING_THE_INITIAL_GROUPS
            end
        end
    endcase
end
```

```

end
if (data[z1] == 1'b1) begin
    implicants[ci][0] <= {1'b0, z1[4]};
    implicants[ci][1] <= {1'b0, z1[3]};
    implicants[ci][2] <= {1'b0, z1[2]};
    implicants[ci][3] <= {1'b0, z1[1]};
    implicants[ci][4] <= {1'b0, z1[0]};
    implicants[ci][5] <= 2'b00;
    ci <= ci+1;
end
z1 <= z1 + 1;
end
FILLING_THE_INITIAL_GROUP: begin
    cil2 <= ci;
    if (l2 < ci) begin
        cntr2 = 0;
        for(z2 = 0; z2 < 5; z2=z2+1) begin//Убрать в отдельное состояние
            if (implicants[l2][z2] == 2'b01)
                cntr2 = cntr2+1;
        end
        groups0 [cntr2] [cn[0][cntr2]][0] <= implicants[l2][0];
        groups0 [cntr2] [cn[0][cntr2]][1] <= implicants[l2][1];
        groups0 [cntr2] [cn[0][cntr2]][2] <= implicants[l2][2];
        groups0 [cntr2] [cn[0][cntr2]][3] <= implicants[l2][3];
        groups0 [cntr2] [cn[0][cntr2]][4] <= implicants[l2][4];
        groups0 [cntr2] [cn[0][cntr2]][5] <= implicants[l2][5];
        //c[l2] <= cntr2;
        cn[0][cntr2] <= cn[0][cntr2] + 1;
        l2 <= l2 + 1;
    end
    else begin
        state <= state + 1;    //COMPARE_FOR_MERGING
    end
end
COMPARE_FOR_MERGING: begin //compare two implicants to understand if can we
merge them
    if (j < cn[m1][cmp_d] && i < cn[m1][cmp_u]) begin
        case(m1)
        0: begin
            for (z3 = 0; z3 < 5; z3=z3+1) begin
                if(groups0[cmp_u][i][z3] != groups0[cmp_d][j][z3])begin
                    local_count = local_count + 1;
                    t_pos = z3;
                end
            end
        end
        1: begin
            for (z3 = 0; z3 < 5; z3=z3+1) begin
                if(groups1[cmp_u][i][z3] != groups1[cmp_d][j][z3])begin
                    local_count = local_count + 1;
                    t_pos = z3;
                end
            end
        end
        2: begin
            for (z3 = 0; z3 < 5; z3=z3+1) begin
                if(groups2[cmp_u][i][z3] != groups2[cmp_d][j][z3])begin
                    local_count = local_count + 1;
                    t_pos = z3;
                end
            end
        end
    end
end
end
end
end

```

```

3: begin
  for (z3 = 0; z3 < 5; z3=z3+1) begin
    if(groups3[cmp_u][i][z3] != groups3[cmp_d][j][z3])begin
      local_count = local_count + 1;
      t_pos = z3;
    end
  end
end
4: begin
  for (z3 = 0; z3 < 5; z3=z3+1) begin
    if(groups4[cmp_u][i][z3] != groups4[cmp_d][j][z3])begin
      local_count = local_count + 1;
      t_pos = z3;
    end
  end
end
5: begin
  for (z3 = 0; z3 < 5; z3=z3+1) begin
    if(groups5[cmp_u][i][z3] != groups5[cmp_d][j][z3])begin
      local_count = local_count + 1;
      t_pos = z3;
    end
  end
end
endcase
if (local_count == 1)begin
  cmp_out <= t_pos;
end
else begin
  cmp_out <= 3'b111;
end
local_count = 0;
end
else begin
  cmp_out <= 3'b111;
end
state <= state + 1;//MERGING
end

MERGING: begin
  if (cmp_u < 5) begin
    if (i < cn[ml_copy1][cmp_u]) begin
      if (j < cn[ml_copy1][cmp_d]) begin
        if (cmp_out < 7) begin
          cf <= 1;
          case(ml_copy1)
            0: begin
              local[0] <= groups0[cmp_u][i][0]; local[1] <=
groups0[cmp_u][i][1]; local[2] <= groups0[cmp_u][i][2];
              local[3] <= groups0[cmp_u][i][3]; local[4] <=
groups0[cmp_u][i][4];
              groups0[cmp_u][i][5] <= 2'b11;
              groups0[cmp_d][j][5] <= 2'b11;
            end
            1: begin
              local[0] <= groups1[cmp_u][i][0]; local[1] <=
groups1[cmp_u][i][1]; local[2] <= groups1[cmp_u][i][2];
              local[3] <= groups1[cmp_u][i][3]; local[4] <=
groups1[cmp_u][i][4];
              groups1[cmp_u][i][5] <= 2'b11;
              groups1[cmp_d][j][5] <= 2'b11;
            end
          endcase
        end
      end
    end
  end
end

```

```

                2: begin
                    local[0] <= groups2[cmp_u][i][0]; local[1] <=
groups2[cmp_u][i][1]; local[2] <= groups2[cmp_u][i][2];
                    local[3] <= groups2[cmp_u][i][3]; local[4] <=
groups2[cmp_u][i][4];
                    groups2[cmp_u][i][5] <= 2'b11;
                    groups2[cmp_d][j][5] <= 2'b11;
                end
                3: begin
                    local[0] <= groups3[cmp_u][i][0]; local[1] <=
groups3[cmp_u][i][1]; local[2] <= groups3[cmp_u][i][2];
                    local[3] <= groups3[cmp_u][i][3]; local[4] <=
groups3[cmp_u][i][4];
                    groups3[cmp_u][i][5] <= 2'b11;
                    groups3[cmp_d][j][5] <= 2'b11;
                end
                4: begin
                    local[0] <= groups4[cmp_u][i][0]; local[1] <=
groups4[cmp_u][i][1]; local[2] <= groups4[cmp_u][i][2];
                    local[3] <= groups4[cmp_u][i][3]; local[4] <=
groups4[cmp_u][i][4];
                    groups4[cmp_u][i][5] <= 2'b11;
                    groups4[cmp_d][j][5] <= 2'b11;
                end
            endcase
            local[cmp_out] <= 2'b10;
            local[5] <= 0;
            state <= state + 1; //GET_COUNT_OF_1_IN_LOCAL
        end
        else begin
            state <= state + 4; //ITERATTION_J
        end
    end
    else begin
        state <= state + 5; //ITERATTION_I
    end
    else begin
        state <= state + 6; //ITERATTION_CMP
    end
end
else begin
    cmp_u <= 0;
    ml <= ml + 1;
    ml_copy1 <= ml_copy1 + 1;
    ml_copy2 <= ml_copy2 + 1;
    ml_copy3 <= ml_copy3 + 1;
    if (cf == 0) begin
        state <= state + 7; //FIND_SIMPLE_IMPLICANT
    end
    cf <= 0;
end
end
GET_COUNT_OF_1_IN_LOCAL: begin
    _c = 0;
    for(z5 = 0; z5 < 5; z5=z5+1) begin
        if (local[z5] == 2'b01)
            _c = _c+1;
    end
    wf <= 1;
    state <= state + 1; //SET_MERGED_VAL_IN_NEXT_COLUMN
end

```

### Продолжение Листинга Б.3

```
SET_MERGED_VAL_IN_NEXT_COLUMN: begin
  if (p < cn[ml_copy2 + 1][_c])begin //Check if we already have this new
implicant in table
    state <= state + 1;//COMPARE_IMPLICANTS
  end
  else begin //If we don't have it - write to table
    if (wf) begin
      case(ml_copy2)
        0: begin
          groups1[_c][cn[1][_c]][0] <= local[0];
groups1[_c][cn[1][_c]][1] <= local[1]; groups1[_c][cn[1][_c]][2] <= local[2];
          groups1[_c][cn[1][_c]][3] <= local[3];
groups1[_c][cn[1][_c]][4] <= local[4]; groups1[_c][cn[1][_c]][5] = local[5];
        end
        1: begin
          groups2[_c][cn[2][_c]][0] <= local[0];
groups2[_c][cn[2][_c]][1] <= local[1]; groups2[_c][cn[2][_c]][2] <= local[2];
          groups2[_c][cn[2][_c]][3] <= local[3];
groups2[_c][cn[2][_c]][4] <= local[4]; groups2[_c][cn[2][_c]][5] = local[5];
        end
        2: begin
          groups3[_c][cn[3][_c]][0] <= local[0];
groups3[_c][cn[3][_c]][1] <= local[1]; groups3[_c][cn[3][_c]][2] <= local[2];
          groups3[_c][cn[3][_c]][3] <= local[3];
groups3[_c][cn[3][_c]][4] <= local[4]; groups3[_c][cn[3][_c]][5] = local[5];
        end
        3: begin
          groups4[_c][cn[4][_c]][0] <= local[0];
groups4[_c][cn[4][_c]][1] <= local[1]; groups4[_c][cn[4][_c]][2] <= local[2];
          groups4[_c][cn[4][_c]][3] <= local[3];
groups4[_c][cn[4][_c]][4] <= local[4]; groups4[_c][cn[4][_c]][5] = local[5];
        end
        4: begin
          groups5[_c][cn[5][_c]][0] <= local[0];
groups5[_c][cn[5][_c]][1] <= local[1]; groups5[_c][cn[5][_c]][2] <= local[2];
          groups5[_c][cn[5][_c]][3] <= local[3];
groups5[_c][cn[5][_c]][4] <= local[4]; groups5[_c][cn[5][_c]][5] = local[5];
        end
      endcase
      cn[ml_copy2+1][_c] <= cn[ml_copy2+1][_c] + 1;
    end
    state <= state + 2;//ITERATTION_J
    p <= 0;
  end
end
COMPARE_IMPLICANTS: begin
  st7_count = 0;
  case(ml_copy3)
    0: begin
      for (z7 = 0; z7 < 5; z7=z7+1)begin
        if(local[z7] == groups1[_c][p][z7]) begin
          st7_count = st7_count + 1;
        end
      end
    end
    1: begin
      for (z7 = 0; z7 < 5; z7=z7+1)begin
        if(local[z7] == groups2[_c][p][z7]) begin
          st7_count = st7_count + 1;
        end
      end
    end
  end
end
```

*Продолжение Листинга Б.3*

```

2: begin
  for (z7 = 0; z7 < 5; z7=z7+1)begin
    if(local[z7] == groups3[_c][p][z7]) begin
      st7_count = st7_count + 1;
    end
  end
end
3: begin
  for (z7 = 0; z7 < 5; z7=z7+1)begin
    if(local[z7] == groups4[_c][p][z7]) begin
      st7_count = st7_count + 1;
    end
  end
end
4: begin
  for (z7 = 0; z7 < 5; z7=z7+1)begin
    if(local[z7] == groups5[_c][p][z7]) begin
      st7_count = st7_count + 1;
    end
  end
endcase
if (st7_count == 5) begin
  wf <= 0;
end
state <= state - 1;//SET_MERGED_VAL_IN_NEXT_COLUMN
p <= p + 1;
end
ITERATTION_J: begin
  j <= j + 1;
  state <= state - 5;//COMPARE_FOR_MERGING
end
ITERATTION_I: begin
  j <= 0;
  i <= i + 1;
  state <= state - 6;//COMPARE_FOR_MERGING
end
ITERATTION_CMP: begin
  j <= 0;
  i <= 0;
  cmp_u <= cmp_u + 1;
  state <= state - 7;//COMPARE_FOR_MERGING
end
FIND_SIMPLE_IMPLICANTS: begin
  if (i11 <= capacity) begin
    if (j11 <= capacity) begin
      if(z11 < cn[i11][j11]) begin
        case(i11)
          0: begin
            if (groups0[j11][z11][5] == 0) begin
              pi[cpi][0] <= groups0[j11][z11][0]; pi[cpi][1] <=
groups0[j11][z11][1]; pi[cpi][2] <= groups0[j11][z11][2];
              pi[cpi][3] <= groups0[j11][z11][3]; pi[cpi][4] <=
groups0[j11][z11][4]; pi[cpi][5] <= groups0[j11][z11][5];
              cpi <= cpi + 1;
            end
          end
          1: begin
            if (groups1[j11][z11][5] == 0) begin
              pi[cpi][0] <= groups1[j11][z11][0]; pi[cpi][1] <=
groups1[j11][z11][1]; pi[cpi][2] <= groups1[j11][z11][2];

```

```

                                pi[cpi][3] <= groups1[j11][z11][3]; pi[cpi][4] <=
groups1[j11][z11][4]; pi[cpi][5] <= groups1[j11][z11][5];
                                cpi <= cpi + 1;
                                end
                                end
                                2: begin
                                    if (groups2[j11][z11][5] == 0) begin
                                        pi[cpi][0] <= groups2[j11][z11][0]; pi[cpi][1] <=
groups2[j11][z11][1]; pi[cpi][2] <= groups2[j11][z11][2];
                                        pi[cpi][3] <= groups2[j11][z11][3]; pi[cpi][4] <=
groups2[j11][z11][4]; pi[cpi][5] <= groups2[j11][z11][5];
                                        cpi <= cpi + 1;
                                    end
                                end
                                end
                                3: begin
                                    if (groups3[j11][z11][5] == 0) begin
                                        pi[cpi][0] <= groups3[j11][z11][0]; pi[cpi][1] <=
groups3[j11][z11][1]; pi[cpi][2] <= groups3[j11][z11][2];
                                        pi[cpi][3] <= groups3[j11][z11][3]; pi[cpi][4] <=
groups3[j11][z11][4]; pi[cpi][5] <= groups3[j11][z11][5];
                                        cpi <= cpi + 1;
                                    end
                                end
                                end
                                4: begin
                                    if (groups4[j11][z11][5] == 0) begin
                                        pi[cpi][0] <= groups4[j11][z11][0]; pi[cpi][1] <=
groups4[j11][z11][1]; pi[cpi][2] <= groups4[j11][z11][2];
                                        pi[cpi][3] <= groups4[j11][z11][3]; pi[cpi][4] <=
groups4[j11][z11][4]; pi[cpi][5] <= groups4[j11][z11][5];
                                        cpi <= cpi + 1;
                                    end
                                end
                                end
                                5: begin
                                    if (groups5[j11][z11][5] == 0) begin
                                        pi[cpi][0] <= groups5[j11][z11][0]; pi[cpi][1] <=
groups5[j11][z11][1]; pi[cpi][2] <= groups5[j11][z11][2];
                                        pi[cpi][3] <= groups5[j11][z11][3]; pi[cpi][4] <=
groups5[j11][z11][4]; pi[cpi][5] <= groups5[j11][z11][5];
                                        cpi <= cpi + 1;
                                    end
                                end
                                end
                                endcase
                                z11 <= z11 + 1;
                                end
                                else begin
                                    z11 <= 0;
                                    j11 <= j11 + 1;
                                end
                                end
                                else begin
                                    i11 <= i11 + 1;
                                    j11 <= 0;
                                end
                                end
                                end
                                state <= state + 1; //FILL_1_POINTS
                                i11 <= 0;
                                end
                                end
                                FILL_1_POINTS: begin
                                    if (z12 < cil2) begin
                                        p1[cp1][0] <= implicants[z12][0];

```

*Продолжение Листинга Б.3*

```

        p1[cp1][1] <= implicants[z12][1];
        p1[cp1][2] <= implicants[z12][2];
        p1[cp1][3] <= implicants[z12][3];
        p1[cp1][4] <= implicants[z12][4];
        p1[cp1][5] <= 2'b00;
        cp1 <= cp1+1;
        z12 <= z12 + 1;
    end
    else begin
        state <= state + 1; //COMPARE_FOR_Q_TABLE
    end
end
COMPARE_FOR_Q_TABLE: begin
    if (i14 < cp1) begin
        if (j14 < cpi) begin
            for(z14 = 0; z14 < 5; z14 = z14 + 1) begin
                if(!((p1[i14][z14] == pi[j14][z14]) || (pi[j14][z14] == 2)))
begin
                    cmp_flag14 <= 0;
                end
                state <= state + 1; //FILL_QUINE_TABLE;
            end
        else begin
            j14 <= 0;
            i14 <= i14 + 1;
        end
    end
    else begin
        i14 <= 0;
        state <= state + 2; //CORE_IMPLICANT_CHECK
    end
end
FILL_QUINE_TABLE: begin
    if (cmp_flag14) begin
        quine_table[i14][j14] <= 1'b1;
    end
    else begin
        quine_table[i14][j14] <= 1'b0;
    end
    state <= state - 1; //COMPARE_FOR_Q_TABLE
    cmp_flag14 <= 1;
    j14 <= j14 + 1;
end
CORE_IMPLICANT_CHECK: begin
    if (i16 < cp1) begin
        if (z16 < cpi) begin
            if (quine_table[i16][z16] == 1) begin
                c16 = c16 + 1;
                pos <= z16;
            end
            z16 <= z16 + 1;
        end
        else begin
            z16 <= 0;
            if (c16 != 1)
                pos <= 5'b11111;
            state <= state + 1; //FIND_CORE_IMPLICANTS
        end
    end
    else begin
        i16 <= 0;

```



### Продолжение Листинга Б.3

```
        state <= state + 2; //INDICATE_COVERED_LINES
    end
end
FIND_CORE_IMPLICANTS: begin
    if (pos != 5'b11111) begin
        pi[pos][5] <= 3;
    end
    i16 <= i16 + 1;
    c16 = 0;
    state <= state - 1; //CORE_IMPLICANT_CHECK
end
INDICATE_COVERED_LINES: begin
    if(i17 < cpi) begin
        if (pi[i17][5] == 3) begin
            if (z17 < cp1) begin
                if(quine_table[z17][i17] == 1) begin
                    p1[z17][5] <= 3;
                end
                z17 <= z17 + 1;
            end
        else begin
            z17 <= 0;
            i17 <= i17 + 1;
        end
    end
    else begin
        i17 <= i17 + 1;
    end
end
    else begin
        i17 <= 0;
        state <= state + 1; //CREATE_PETRICK_TABLE
    end
end
CREATE_PETRICK_TABLE: begin
    if (z18 < cp1) begin
        if(p1[z18][5] == 0) begin
            if (l18 < cpi) begin
                petrick_table0[clpt[0]][l18] <= quine_table[z18][l18];
                l18 <= l18 + 1;
            end
        else begin
            z18 <= z18 + 1;
            l18 <= 0;
            clpt[0] <= clpt[0] + 1;
        end
    end
    else
        z18 <= z18 + 1;
    end
    else begin
        state <= state + 1;
    end
end
PETRICK_METHODS_CALCULATIONS: begin
    if(clpt[0] > 0) begin
        if (i19 < cpi) begin
            if (z19 < clpt[p1]) begin
                case (p1)
                0: begin
                    if(petrick_table0[z19][i19] == 1)begin
                        clc <= clc + 1;
                    end
                end
            end
        end
    end
end
```

```

        end
    end
    1: begin
        if(petrick_table1[z19][i19] == 1)begin
            clc <= clc + 1;
        end
    end
    2: begin
        if(petrick_table2[z19][i19] == 1)begin
            clc <= clc + 1;
        end
    end
    3: begin
        if(petrick_table3[z19][i19] == 1)begin
            clc <= clc + 1;
        end
    end
    4: begin
        if(petrick_table4[z19][i19] == 1)begin
            clc <= clc + 1;
        end
    end
    5: begin
        if(petrick_table5[z19][i19] == 1)begin
            clc <= clc + 1;
        end
    end
    6: begin
        if(petrick_table6[z19][i19] == 1)begin
            clc <= clc + 1;
        end
    end
    7: begin
        if(petrick_table7[z19][i19] == 1)begin
            clc <= clc + 1;
        end
    end
    8: begin
        if(petrick_table8[z19][i19] == 1)begin
            clc <= clc + 1;
        end
    end
    endcase
    z19 <= z19 + 1;
end
else begin
    z19 <= 0;
    if (clc > max_clc) begin
        max_clc <= clc;
        pei <= i19;
    end
    clc <= 0;
    i19 <= i19 + 1;
end
end
else begin
    pi[pei][5] <= 3;
    if (l19 < clpt[pl]) begin
        case (pl)
        0: begin
            if(petrick_table0[l19][pei] != 1) begin
                if (t19 < cpi) begin

```

```
petrick_table1[clpt[1]][t19] <=
petrick_table0[l19][t19];
    t19 <= t19 + 1;
    end
    else begin
        l19 <= l19 + 1;
        t19 <= 0;
        clpt[1] <= clpt[1] + 1;
    end
end
else
    l19 <= l19 + 1;
end
1: begin
    if(petrick_table1[l19][pei] != 1) begin
        if (t19 < cpi) begin
            petrick_table2[clpt[2]][t19] <=
petrick_table1[l19][t19];
            t19 <= t19 + 1;
            end
            else begin
                l19 <= l19 + 1;
                t19 <= 0;
                clpt[2] <= clpt[2] + 1;
            end
        end
        else
            l19 <= l19 + 1;
        end
    2: begin
        if(petrick_table2[l19][pei] != 1) begin
            if (t19 < cpi) begin
                petrick_table3[clpt[3]][t19] <=
petrick_table2[l19][t19];
                t19 <= t19 + 1;
                end
                else begin
                    l19 <= l19 + 1;
                    t19 <= 0;
                    clpt[3] <= clpt[3] + 1;
                end
            end
            else
                l19 <= l19 + 1;
            end
        3: begin
            if(petrick_table3[l19][pei] != 1) begin
                if (t19 < cpi) begin
                    petrick_table4[clpt[4]][t19] <=
petrick_table3[l19][t19];
                    t19 <= t19 + 1;
                    end
                    else begin
                        l19 <= l19 + 1;
                        t19 <= 0;
                        clpt[4] <= clpt[4] + 1;
                    end
                end
                else
                    l19 <= l19 + 1;
                end
            4: begin
```

```
        if(petrick_table4[l19][pei] != 1) begin
            if (t19 < cpi) begin
                petrick_table5[clpt[5]][t19] <=
petrick_table4[l19][t19];
                t19 <= t19 + 1;
            end
            else begin
                l19 <= l19 + 1;
                t19 <= 0;
                clpt[5] <= clpt[5] + 1;
            end
        end
        else
            l19 <= l19 + 1;
    end
5: begin
    if(petrick_table5[l19][pei] != 1) begin
        if (t19 < cpi) begin
            petrick_table6[clpt[6]][t19] <=
petrick_table5[l19][t19];
            t19 <= t19 + 1;
        end
        else begin
            l19 <= l19 + 1;
            t19 <= 0;
            clpt[6] <= clpt[6] + 1;
        end
    end
    else
        l19 <= l19 + 1;
    end
6: begin
    if(petrick_table6[l19][pei] != 1) begin
        if (t19 < cpi) begin
            petrick_table7[clpt[7]][t19] <=
petrick_table6[l19][t19];
            t19 <= t19 + 1;
        end
        else begin
            l19 <= l19 + 1;
            t19 <= 0;
            clpt[7] <= clpt[7] + 1;
        end
    end
    else
        l19 <= l19 + 1;
    end
7: begin
    if(petrick_table7[l19][pei] != 1) begin
        if (t19 < cpi) begin
            petrick_table8[clpt[8]][t19] <=
petrick_table7[l19][t19];
            t19 <= t19 + 1;
        end
        else begin
            l19 <= l19 + 1;
            t19 <= 0;
            clpt[8] <= clpt[8] + 1;
        end
    end
    else
        l19 <= l19 + 1;
    end
```

*Продолжение Листинга Б.3*

```

        end
        8: begin
            l19 <= l19 + 1;
        end
    endcase
end
else begin
    l19 <= 0;
    if (clpt[p1 + 1] == 0)begin
        state <= state + 1;
    end
    p1 <= p1 + 1;
    max_clc <= 0;
    i19 <= 0;
end
end
end
else begin
    state <= state + 1;
end
end
WRITE_RESULT: begin
    if (z20 < cpi) begin
        if(pi[z20][5] == 3) begin
            result_reg[res_count][0] <= pi[z20][0]; result_reg[res_count][1]
<= pi[z20][1];
            result_reg[res_count][2] <= pi[z20][2]; result_reg[res_count][3]
<= pi[z20][3];
            result_reg[res_count][4] <= pi[z20][4];
            res_count <= res_count + 1;
        end
        z20 <= z20 + 1;
    end
    else
        state <= state + 1;
    end
    READY_RESULT_FLAG: begin
        ready_result <= 1;
        state <= state + 1;
    end
    SEND_RESULT: begin
        ready_result <= 0;
        if(i22 < res_count) begin
            result[9:8] <= result_reg[i22][0]; result[7:6] <=
result_reg[i22][1];
            result[5:4] <= result_reg[i22][2]; result[3:2] <=
result_reg[i22][3];
            result[1:0] <= result_reg[i22][4];
            i22 <= i22 + 1;
        end
        else begin
            result_end <= 1;
            state <= RESET;
        end
    end
end
RESET: begin
    result_end <= 0;
    ready_result <= 0;
    result <= 0;
    res_count <= 0;
    ci <= 5'b000000;
    i <= 0; j <= 0;
end
```

*Продолжение Листинга Б.3*

```
        for(k = 0; k < 6; k=k+1)begin
            for(x = 0; x < 6; x=x+1)begin
                cn[k][x] = 5'b000000;
            end
        end
    end
    cf <= 0;
    ml <= 0;
    ml_copy1 <= 0;
    ml_copy2 <= 0;
    ml_copy3 <= 0;
    cmp_u <= 0;
    p <= 0;
    local_count = 0;
    cpi <= 0;
    i11 <= 0;
    j11 <= 0;
    cpl <= 0;
    i14 <= 0;
    j14 <= 0;
    cmp_flag14 <= 1;
    i16 <= 0; z16 <= 0; pos <= 0; c16 = 0;
    i17 <= 0;
    pl <= 0;
    for (t = 0; t < 29; t = t + 1) begin
        clpt[t] <= 6'b0000000;
    end
    pei <= 0; clc <= 0; max_clc <= 0; i19 <= 0;
    i22 <= 0;
    state <= 0;
    k = 0; t = 0; z2 = 0; z1 = 0; z3 = 0; l2 <= 0; z5 = 0; z7 = 0; z11 <= 0;
z12 <= 0; z14 = 0;
    cmp_out <= 0;
    local[0] <= 0; local[1] <= 0; local[2] <= 0;
    local[3] <= 0; local[4] <= 0; local[5] <= 0;
    t_pos = 0;
    _c = 0;
    z17 = 0; z18 <= 0; l18 <= 0; z19 <= 0; l19 <= 0; t19 <= 0; z20 <= 0;
    end
endcase
end
endmodule
```

## Приложение Б.4

### Модуль Transmitter

*Листинг Б.4 — Реализация модуля Transmitter*

```
module Transmitter(
    input [9:0] result, [5:0] res_count, [2:0] capacity, [2:0] error_reg,
    input empty, send, clk,
    output t, reg ready_next
);

// UART_TRANSMITTER
reg [7:0] data_ut;
reg start;
wire rNext;

// ASCII_CODER
reg [4:0] code;
wire [7:0] ascii_code;

reg [5:0] cnt;
reg [3:0] p;
reg [3:0] z;
reg [5:0] rc;
integer state;

parameter WAIT_SEND = 0;
parameter SEND_LETTER = 1;
parameter WAIT_rNext = 2;
parameter SEND_ERROR = 3;
parameter SEND_NUM_ERROR = 4;
parameter SEND_CR = 5;
parameter END = 6;
parameter PART_END = 7;
parameter WAIT_NEXT_PART = 8;
parameter SEND_CR_ER = 9;

initial begin
    cnt = 0;
    p = 0;
    state = 0;
    start = 0;
    rc = 1;
    ready_next = 0;
end

always@(posedge clk)begin
    case(state)
        WAIT_SEND: begin
            ready_next <= 0;
            start <= 0;
            if(send) begin
                if (error_reg == 0) begin
                    p <= (10 - 2*(5 - capacity)) - 1;
                    state <= state + 1;
                end
            end
            else begin
                state <= state + 3;//SEND_ERROR
            end
        end
    end
end
```

#### Продолжение Листинга Б.4

```
SEND_LETTER: begin
  if (cnt < capacity) begin
    case(cnt)
      0: begin
        if(result[p] == 0 && result[p-1] == 0) begin
          code <= 5'b01010;
          start <= 1;
          state <= state + 1;
        end
        else if(result[p-1] == 1) begin
          code <= 5'b11010;
          start <= 1;
          state <= state + 1;
        end
      end
    end
    1: begin
      if(result[p-2] == 0 && result[p-3] == 0) begin
        code <= 5'b01011;
        start <= 1;
        state <= state + 1;
      end
      else if(result[p-3] == 1) begin
        code <= 5'b11011;
        start <= 1;
        state <= state + 1;
      end
    end
    2: begin
      if(result[p-4] == 0 && result[p-5] == 0) begin
        code <= 5'b01100;
        start <= 1;
        state <= state + 1;
      end
      else if(result[p-5] == 1) begin
        code <= 5'b11100;
        start <= 1;
        state <= state + 1;
      end
    end
    3: begin
      if(result[p-6] == 0 && result[p-7] == 0) begin
        code <= 5'b01101;
        start <= 1;
        state <= state + 1;
      end
      else if(result[p-7] == 1) begin
        code <= 5'b11101;
        start <= 1;
        state <= state + 1;
      end
    end
    4: begin
      if(result[p-8] == 0 && result[p-9] == 0) begin
        code <= 5'b01110;
        start <= 1;
        state <= state + 1;
      end
      else if(result[p-9] == 1) begin
        code <= 5'b11110;
        start <= 1;
        state <= state + 1;
      end
    end
  end
end
```



```
        end
        endcase
        cnt <= cnt + 1;
    end
    else begin
        cnt <= 0;
        if (rc != res_count) begin
            code <= 5'b10001;
            start <= 1;
            state <= PART_END;
            rc <= rc + 1;
        end
        else begin
            state <= SEND_CR;
            rc <= 1;
        end
    end
end
end
WAIT_rNext: begin
    start <= 0;
    if(rNext) begin
        state <= state - 1;
    end
end
SEND_ERROR: begin
    code <= 5'b01110;
    start <= 1;
    state <= state + 1;
end
SEND_NUM_ERROR: begin
    if(rNext) begin
        case(error_reg)
            1: begin
                code <= 5'b00001;
            end
            2: begin
                code <= 5'b00010;
            end
        endcase
        state <= SEND_CR_ER;
    end
end
SEND_CR_ER: begin
    if (rNext) begin
        code <= 5'b10000;
        state <= END;
    end
end
SEND_CR: begin
    code <= 5'b10000;
    start <= 1;
    state <= state + 1;
end
END: begin
    if (rNext) begin
        start <= 0;
        state <= 0;
        ready_next <= 1;
    end
end
PART_END: begin
    start <= 0;
```

*Продолжение Листинга Б.4*

```
        if (rNext) begin
            ready_next <= 1;
            state <= state + 1;
        end
    end
    WAIT_NEXT_PART: begin
        ready_next <= 0;
        if (send) begin
            state <= SEND_LETTER;
        end
    end
endcase
end

UART_TRANSMITTER ut (clk, start, ascii_code, t, rNext);
ASCII_CODER ac (code, ascii_code);

endmodule
```

## Приложение Б.5

### Модуль UART\_TRANSMITTER

*Листинг Б.5 — Реализация модуля UART\_TRANSMITTER*

```
module UART_TRANSMITTER
#(parameter BRate = 9600)
(input CLK, start, [7:0] data,
output reg t, rNext);

localparam
    NEXCLK = 100_000_000,
    period = NEXCLK/BRate,
    bit0 = period,
    bit1 = 2*period,
    bit2 = 3*period,
    bit3 = 4*period,
    bit4 = 5*period,
    bit5 = 6*period,
    bit6 = 7*period,
    bit7 = 8*period,
    stop = 9*period,
    ending = 15*period;

reg [$clog2(ending):0] state;

initial
begin
    t = 0;
    rNext = 0;
    state = 0;
end

always @(posedge CLK) begin
    case (state)
        0: begin
            t <= 1;
            rNext <= 0;
            if (start) state <= 1;
        end
        1: begin
            t <= 0;
            state <= state + 1;
        end
        bit0: begin
            t <= data[0];
            state <= state + 1;
        end
        bit1: begin
            t <= data[1];
            state <= state + 1;
        end
        bit2: begin
            t <= data[2];
            state <= state + 1;
        end
        bit3: begin
            t <= data[3];
            state <= state + 1;
        end
        bit4: begin
```

*Продолжение Листинга Б.5*

```
        t <= data[4];
        state <= state + 1;
    end
    bit5: begin
        t <= data[5];
        state <= state + 1;
    end
    bit6: begin
        t <= data[6];
        state <= state + 1;
    end
    bit7: begin
        t <= data[7];
        state <= state + 1;
    end
    stop: begin
        t <= 1;
        state <= state + 1;
    end
    ending: begin
        state <= 0;
        rNext <= 1;
    end
    default: state <= state + 1;
endcase
end
endmodule
```

## Приложение Б.6

### Модуль ASCII\_CODER

*Листинг Б.6 — Реализация модуля ASCII\_CODER*

```
module ASCII_CODER (code, ascii_code);
input [4:0] code;
output reg [7:0] ascii_code;

always @(*) begin
    case (code)

        5'b00001: ascii_code <= 8'h31;//1
        5'b00010: ascii_code <= 8'h32;//2
        5'b00011: ascii_code <= 8'h33;//3
        5'b00100: ascii_code <= 8'h34;//4
        5'b00101: ascii_code <= 8'h35;//5

        5'b11010: ascii_code <= 8'h41;//A
        5'b11011: ascii_code <= 8'h42;//B
        5'b11100: ascii_code <= 8'h43;//C
        5'b11101: ascii_code <= 8'h44;//D
        5'b11110: ascii_code <= 8'h45;//E

        5'b01010: ascii_code <= 8'h61;//a
        5'b01011: ascii_code <= 8'h62;//b
        5'b01100: ascii_code <= 8'h63;//c
        5'b01101: ascii_code <= 8'h64;//d
        5'b01110: ascii_code <= 8'h65;//e

        5'b10001: ascii_code <= 8'h2b;//+
        5'b10000: ascii_code <= 8'h0d;//перенос картекти
    endcase
end
endmodule
```

## Приложение Б.7

### Модуль Reciever

Листинг Б.7 — Реализация модуля Reciever

```
module Reciever(
    input
        r,
        clk,
    output
        reg [0:31] data,
        reg [2:0] capacity,
        reg [2:0] error_reg,
        reg write
);

parameter s_capacity = 0;
parameter s_data = 1;

//UART_RECIEVER
wire R_O;
wire [7:0] received_data;
//ASCII_DECODER
wire [0:3] code; //Полученное число

integer c;
integer state;
initial begin
    data = 0;
    capacity = 0;
    error_reg = 0;
    state = 0;
    c = 0;
    write = 0;
end

always@(posedge clk) begin
    if(R_O) begin
        case(state)
            s_capacity: begin
                if (code == 4'b1110) begin
                    error_reg <= 4'b0010; //Недопустимое количество чисел |
Incorrect count of numbers
                    write <= 1; //загрузить результат в fifo
                end
                else if (code == 4'b1000 || code < 3) begin
                    error_reg <= 4'b0001; //Недопустимое число | Incorrect
number
                    state <= state + 1;
                end
                else if (code <= 5 && code >= 3)begin
                    capacity <= code;
                    state <= state + 1;
                end
            end
            s_data: begin
                if (code == 4'b1110) begin
                    if(c != 2**capacity) begin
                        if(error_reg == 0) begin
                            error_reg <= 4'b0010; //Недопустимое количество
чисел | Incorrect count of numbers

```

*Продолжение Листинга Б.7*

```

        end
        end
        write <= 1;
        state <= 0;
        c <= 0;
    end
    else if (code > 1 && code < 9) begin
        if(error_reg == 0) begin
            error_reg <= 4'b0001; //Недопустимое число | Incorrect
number
        end
        c <= c + 1;
    end
    else if (code == 0 || code == 1) begin
        if(c < 32)
            data[c] <= code[3];
        c <= c + 1;
    end
    end
    endcase
end
else if (write) begin
    write <= 0;
    data <= 0;
    capacity <= 0;
    error_reg <= 0;
end
end

UART_RECIEVER ur (r, clk, R_O, received_data);
ASCII_DECODER ad (received_data, code);

endmodule
```

## Приложение Б.8

### Модуль UART\_RECIEVER

*Листинг Б.8 — Реализация модуля UART\_RECIEVER*

```
module UART_RECIEVER
#(parameter BRate = 9600)
(input r, CLK,
output reg R_O, reg [7:0] received_data);

localparam
    NEXCLK = 100_000_000,
    period = NEXCLK/BRate/2,
    waiting = period,
    bit0 = 3*period,
    bit1 = 5*period,
    bit2 = 7*period,
    bit3 = 9*period,
    bit4 = 11*period,
    bit5 = 13*period,
    bit6 = 15*period,
    bit7 = 17*period,
    stop = 19*period;

reg [$clog2(stop):0] state;

initial
begin
    R_O = 0;
    state = 0;
    received_data = 0;
end

always @(posedge CLK) begin
    if (state == 0) begin
        R_O <= 0;
        state <= state + 1;
    end

    else if (state <= waiting)begin
        if (~r) state <= state + 1;
    end
    else case (state)
        bit0: begin
            received_data[0] <= r;
            state <= state + 1;
        end

        bit1: begin
            received_data[1] <= r;
            state <= state + 1;
        end

        bit2: begin
            received_data[2] <= r;
            state <= state + 1;
        end

        bit3: begin
            received_data[3] <= r;
            state <= state + 1;
        end
    end
end
```



*Продолжение Листинга Б.8*

```
end

bit4: begin
    received_data[4] <= r;
    state <= state + 1;
end

bit5: begin
    received_data[5] <= r;
    state <= state + 1;
end

bit6: begin
    received_data[6] <= r;
    state <= state + 1;
end

bit7: begin
    received_data[7] <= r;
    state <= state + 1;
end

stop: begin
    R_O <= 1;
    state <= 0;
end

default: state <= state + 1;
endcase
end
endmodule
```

## Приложение Б.9

### Модуль ASCII\_DECODER

*Листинг Б.9 — Реализация модуля ASCII\_DECODER*

```
module ASCII_DECODER(ascii_code, code);
input [7:0] ascii_code;
output reg [3:0] code;

always @(*) begin
    case (ascii_code)
        8'h30: code <= 4'b0000; // 0
        8'h31: code <= 4'b0001; // 1

        8'h33: code <= 4'b0011; // 3
        8'h34: code <= 4'b0100; // 4
        8'h35: code <= 4'b0101; // 5

        8'h3b: code <= 4'b1110; // ;
        8'h0a: code <= 4'b1101; // LF
        8'h0d: code <= 4'b1100; // CR

        default: code <= 4'b1000;
    endcase
end
endmodule
```

## Приложение Б.10

### Модуль REVERSE\_ASCII\_CODER

*Листинг Б.10 — Реализация модуля REVERSE\_ASCII\_CODER*

```
module REVERSE_ASCII_CODER(code, ascii_code);
input [3:0] code;
output reg [7:0] ascii_code;

always @(*) begin
    case (code)
        4'b0000: ascii_code <= 8'h30; //0
        4'b0001: ascii_code <= 8'h31; //1

        4'b0011: ascii_code <= 8'h33; //3
        4'b0100: ascii_code <= 8'h34; //4
        4'b0101: ascii_code <= 8'h35; //5

        4'b1110: ascii_code <= 8'h3b; //;
        4'b1101: ascii_code <= 8'h0a; //LF
        4'b1100: ascii_code <= 8'h0d; //CR

        4'b1111: ascii_code <= 8'h46;
        default: ascii_code <= 8'h0d;
    endcase
end

endmodule
```

## Приложение Б.11

### Модуль REVERSE\_ASCII\_CODER

*Листинг Б.11 — Реализация модуля REVERSE\_ASCII\_CODER*

```
module REVERSE_ASCII_DECODER(ascii_code, code);
input [7:0] ascii_code;
output reg [4:0] code;

always @(*) begin
    case (ascii_code)
        8'h31 : code <= 5'b000001;//1
        8'h32 : code <= 5'b000010;//2
        8'h33 : code <= 5'b000011;//3
        8'h34 : code <= 5'b000100;//4
        8'h35 : code <= 5'b000101;//5

        8'h41 : code <= 5'b110101;//A
        8'h42 : code <= 5'b110111;//B
        8'h43 : code <= 5'b111001;//C
        8'h44 : code <= 5'b111011;//D
        8'h45 : code <= 5'b111101;//E

        8'h61 : code <= 5'b010101;//a
        8'h62 : code <= 5'b010111;//b
        8'h63 : code <= 5'b011001;//c
        8'h64 : code <= 5'b011011;//d
        8'h65 : code <= 5'b011101;//e

        8'h2b : code <= 5'b100001;//+
        8'h0d : code <= 5'b100000;//перенос картекти
    endcase
end

endmodule
```

## Приложение В

### Модуль top\_module\_test

*Листинг В — Тестовый модуль top\_module\_test*

```
module top_module_test();

integer state;
integer n;
reg [0:35] DATA1;
reg [0:67] DATA2;
reg [0:131] DATA3;
// [31:0] res;

//Работ с TOP_MODULE
wire t;
wire r;
reg clk;

//Работа с UART_TRANSMITTER
wire rNext;
reg start;
wire [0:7] out_symbol;
reg [0:3] out_num;

//Выходы из модуля UART_RECIEVER
wire R_RECIEVER; //Однотактовый сигнал, означающий, что пакет записан в
received_data
wire [7:0] received_data;
wire [4:0] value;

initial begin
    clk = 0;
    state = 0;
    n = 0;
    DATA1 = 36'h301110101;
    DATA2 = 68'h4011111101110111;
    DATA3 = 132'h50111111011101110100000001110111;
end

always #10 begin
    clk = ~clk;
end

always@(posedge clk)
begin
    if (state == 0)
        begin
            if (n == 0)
                begin
                    out_num <= DATA1[0:3];
                    start <= 1;
                    n <= n + 1;
                end
            else if (rNext)
                begin
                    case(n)
                        1: begin
                            out_num <= DATA1[4:7];
                            start <= 1;
                        end
                    endcase
                end
        end
    end
end
```

```
        n <= n + 1;
    end
    2: begin
        out_num <= DATA1[8:11];
        start <= 1;
        n <= n + 1;
    end
    3: begin
        out_num <= DATA1[12:15];
        start <= 1;
        n <= n + 1;
    end
    4: begin
        out_num <= DATA1[16:19];
        start <= 1;
        n <= n + 1;
    end
    5: begin
        out_num <= DATA1[20:23];
        start <= 1;
        n <= n + 1;
    end
    6: begin
        out_num <= DATA1[24:27];
        start <= 1;
        n <= n + 1;
    end
    7: begin
        out_num <= DATA1[28:31];
        start <= 1;
        n <= n + 1;
    end
    8: begin
        out_num <= DATA1[32:35];
        start <= 1;
        n <= n + 1;
    end
    9: begin
        out_num <= 4'b1110;
        start <= 1;
        n <= n + 1;
    end
    10: begin
        n <= 0;
        state <= 1;
        start <= 0;
    end
endcase
    end
else
    begin
        start <= 0;
    end
end
if (state == 1)
    begin
        if (n == 0)
            begin
                out_num <= DATA2[0:3];
                start <= 1;
                n <= n + 1;
            end
        end
    end
end
```

```
else if (rNext)
begin
case(n)
1: begin
out_num <= DATA2[4:7];
start <= 1;
n <= n + 1;
end
2: begin
out_num <= DATA2[8:11];
start <= 1;
n <= n + 1;
end
3: begin
out_num <= DATA2[12:15];
start <= 1;
n <= n + 1;
end
4: begin
out_num <= DATA2[16:19];
start <= 1;
n <= n + 1;
end
5: begin
out_num <= DATA2[20:23];
start <= 1;
n <= n + 1;
end
6: begin
out_num <= DATA2[24:27];
start <= 1;
n <= n + 1;
end
7: begin
out_num <= DATA2[28:31];
start <= 1;
n <= n + 1;
end
8: begin
out_num <= DATA2[32:35];
start <= 1;
n <= n + 1;
end
9: begin
out_num <= DATA2[36:39];
start <= 1;
n <= n + 1;
end
10: begin
out_num <= DATA2[40:43];
start <= 1;
n <= n + 1;
end
11: begin
out_num <= DATA2[44:47];
start <= 1;
n <= n + 1;
end
12: begin
out_num <= DATA2[48:51];
start <= 1;
n <= n + 1;
```

```
end
13: begin
    out_num <= DATA2[52:55];
    start <= 1;
    n <= n + 1;
end
14: begin
    out_num <= DATA2[56:59];
    start <= 1;
    n <= n + 1;
end
15: begin
    out_num <= DATA2[60:63];
    start <= 1;
    n <= n + 1;
end
16: begin
    out_num <= DATA2[64:67];
    start <= 1;
    n <= n + 1;
end
17: begin
    out_num <= 4'b1110;
    start <= 1;
    n <= n + 1;
end
18: begin
    n <= 0;
    state <= 2;
    start <= 0;
end
endcase
end
else
begin
    start <= 0;
end
end
if (state == 2)
begin
    if (n == 0)
begin
    out_num <= DATA3[0:3];
    start <= 1;
    n <= n + 1;
end
    else if (rNext)
begin
    case(n)
    1: begin
        out_num <= DATA3[4:7];
        start <= 1;
        n <= n + 1;
    end
    2: begin
        out_num <= DATA3[8:11];
        start <= 1;
        n <= n + 1;
    end
    3: begin
        out_num <= DATA3[12:15];
        start <= 1;
    end
end
end
```



```
        n <= n + 1;
    end
4: begin
    out_num <= DATA3[16:19];
    start <= 1;
    n <= n + 1;
end
5: begin
    out_num <= DATA3[20:23];
    start <= 1;
    n <= n + 1;
end
6: begin
    out_num <= DATA3[24:27];
    start <= 1;
    n <= n + 1;
end
7: begin
    out_num <= DATA3[28:31];
    start <= 1;
    n <= n + 1;
end
8: begin
    out_num <= DATA3[32:35];
    start <= 1;
    n <= n + 1;
end
9: begin
    out_num <= DATA3[36:39];
    start <= 1;
    n <= n + 1;
end
10: begin
    out_num <= DATA3[40:43];
    start <= 1;
    n <= n + 1;
end
11: begin
    out_num <= DATA3[44:47];
    start <= 1;
    n <= n + 1;
end
12: begin
    out_num <= DATA3[48:51];
    start <= 1;
    n <= n + 1;
end
13: begin
    out_num <= DATA3[52:55];
    start <= 1;
    n <= n + 1;
end
14: begin
    out_num <= DATA3[56:59];
    start <= 1;
    n <= n + 1;
end
15: begin
    out_num <= DATA3[60:63];
    start <= 1;
    n <= n + 1;
end
```

```
16: begin
    out_num <= DATA3[64:67];
    start <= 1;
    n <= n + 1;
end
17: begin
    out_num <= DATA3[68:71];
    start <= 1;
    n <= n + 1;
end
18: begin
    out_num <= DATA3[72:75];
    start <= 1;
    n <= n + 1;
end
19: begin
    out_num <= DATA3[76:79];
    start <= 1;
    n <= n + 1;
end
20: begin
    out_num <= DATA3[80:83];
    start <= 1;
    n <= n + 1;
end
21: begin
    out_num <= DATA3[84:87];
    start <= 1;
    n <= n + 1;
end
22: begin
    out_num <= DATA3[88:91];
    start <= 1;
    n <= n + 1;
end
23: begin
    out_num <= DATA3[92:95];
    start <= 1;
    n <= n + 1;
end
24: begin
    out_num <= DATA3[96:99];
    start <= 1;
    n <= n + 1;
end
25: begin
    out_num <= DATA3[100:103];
    start <= 1;
    n <= n + 1;
end
26: begin
    out_num <= DATA3[104:107];
    start <= 1;
    n <= n + 1;
end
27: begin
    out_num <= DATA3[108:111];
    start <= 1;
    n <= n + 1;
end
28: begin
    out_num <= DATA3[112:115];
```

```
        start <= 1;
        n <= n + 1;
    end
    29: begin
        out_num <= DATA3[116:119];
        start <= 1;
        n <= n + 1;
    end
    30: begin
        out_num <= DATA3[120:123];
        start <= 1;
        n <= n + 1;
    end
    31: begin
        out_num <= DATA3[124:127];
        start <= 1;
        n <= n + 1;
    end
    32: begin
        out_num <= DATA3[128:131];
        start <= 1;
        n <= n + 1;
    end
    33: begin
        out_num <= 4'b1110;
        start <= 1;
        n <= n + 1;
    end
    34: begin
        n <= 0;
        state <= 3;
        start <= 0;
    end
    endcase
end
else
    begin
        start <= 0;
    end
end
if (state == 3)
    begin
        if (n == 0)
            begin
                out_num <= 4'b1110;
                start <= 1;
                n <= n + 1;
            end
        else if (rNext)
            begin
                case(n)
                    1: begin
                        out_num <= 4'b1110;
                        start <= 1;
                        n <= n + 1;
                    end
                    2: begin
                        out_num <= 4'b1101;
                        start <= 1;
                        n <= n + 1;
                    end
                    3: begin
```

```
        out_num <= 4'b1100;
        start <= 1;
        n <= n + 1;
    end
    4: begin
        out_num <= 4'b1111;
        start <= 1;
        n <= n + 1;
    end
    5: begin
        out_num <= 4'b0011;
        start <= 1;
        n <= n + 1;
    end
    6: begin
        out_num <= 4'b0000;
        start <= 1;
        n <= n + 1;
    end
    7: begin
        out_num <= 4'b0001;
        start <= 1;
        n <= n + 1;
    end
    8: begin
        out_num <= 4'b1101;
        start <= 1;
        n <= n + 1;
    end
    9: begin
        out_num <= 4'b1110;
        start <= 1;
        n <= n + 1;
    end
    10: begin
        n <= 0;
        state <= 0;
        start <= 1;
    end
endcase
    end
else
    begin
        start <= 0;
    end
end
end

TOP_MODULE tm (.clk(clk), .r(t), .t(r));

UART_TRANSMITTER #(9600) ut (.CLK(clk), .start(start), .data(out_symbol), .t(t),
.rNext(rNext));
REVERSE_ASCII_CODER rac (.code(out_num), .ascii_code(out_symbol));

UART_RECIEVER #(9600) ur (.r(r), .CLK(clk), .R_O(R_RECIEVER),
.received_data(received_data));
REVERSE_ASCII_DECODER rad (.ascii_code(received_data), .code(value));

endmodule
```

**ОТЗЫВ**  
на курсовую работу  
по дисциплине «Схемотехника устройств компьютерных систем»

Студент(ка) Паращенко Федор Дмитриевич Группа ИБО-04-21

Характеристика курсовой работы

Критерий	Да	Нет	Не полностью
1. Соответствие содержания курсовой работы указанной теме	+		
2. Соответствие курсовой работы заданию	+		
3. Соответствие рекомендациям по оформлению текста, таблиц, рисунков и пр.			+
4. Полнота выполнения всех пунктов задания	+		
5. Логичность и системность содержания курсовой работы	+		
6. Отсутствие фактических грубых ошибок	+		

Рекомендуемая оценка: удовлетворительно, хорошо, отлично



Подпись руководителя

Потехин Д.С.

(ФИО руководителя)

«23» декабря 2023 г.