Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления
Кафедра Интеллектуальных информационных технологий

ОТЧЁТ

по расчетной работе

Выполнили: Г. Г. Говор
М. В. Балюк
М. А. Фролов

Студенты группы
421702

Проверил: Н. В. Зотов

Минск 2025

СОДЕРЖАНИЕ

1 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Вариант 9. Размещение компонентов на печатной плате.

Постановка задачи: Компания-разработчик систем “УмныйДом-2025” проектирует печатную плату для микроконтроллерного устройства управления. На плате размещаются ключевые электронные компоненты: микропроцессор, различные микросхемы памяти, разъемы, резисторы, конденсаторы. Электрические соединения образуют сложную сеть дорожек, которые должны учитывать технологические ограничения (минимальные расстояния между компонентами и проводниками, используемые корпуса, тепловые и электромагнитные требования). Важно определить, возможно ли расположить все соединения и элементы на одной плате без пересечений дорожек, сколько слоев потребуется для разводки, как оптимально сгруппировать компоненты с точки зрения монтажа, обслуживания и безопасности.

Пример:

Компоненты:

- Микропроцессор U1 (ARM Cortex-M4, QFP-64, 8×8 мм);
- Flash память U2 (SOIC-8, 5×4 мм);
- EEPROM U3 (SOIC-8, 5×4 мм);
- RAM U4 (TSOP-28, 11×4 мм);
- SD-карта: U5 (разъем, 15×11 мм);
- Буферные памяти U6-U9 (SOT-23, 3×3 мм);
- Резисторы R1–R12 (2×1 мм);
- Конденсаторы C1–C6 (2×1 мм).

Требования к соединениям:

- 15 дорожек с необходимыми трассировками (питание, шины данных, адреса, спецсигналы);
- минимальное расстояние между компонентами: 0.5 мм;
- ширина дорожки: 0.2 мм; между дорожками: 0.2 мм; диаметр переходного отверстия (via): 0.3 мм.

Тепловые ограничения:

- U1 выделяет до 200 мВт тепла, требуется заливка;
- минимальное расстояние от U1 до термочувствительных компонентов: 5 мм.

Электромагнитные ограничения:

- высокочастотные сигналы (CLK_EXT) — экранировать; длина до 15 мм.

Входные данные: Таблица компонентов с размерами, типами корпусов, требованиями по соединениям; список электрических дорожек с назначением и ограничениями, тепловые и электромагнитные параметры.

Требуется:

1. Построить граф соединений (вершины — компоненты, рёбра — электрические связи, каждая с типом сигнала и требованиями по разводке).

2. Проверить планарность графа (возможно ли разместить все соединения на одном слое без пересечений).
3. Если планарность невозможна — вычислить минимальное количество пересечений/слоев для полного размещения.
4. Осуществить оптимальную раскладку компонентов в пределах площади платы 60×40 мм с учетом тепловых и монтажных ограничений, доступности для обслуживания, группировок по функционалу.

2 РЕАЛИЗАЦИЯ БАЗЫ ЗНАНИЙ

2.1 Описание разработанной предметной области

Для формализации задач на языке SCs были выделены следующие ключевые элементы:

- **Абсолютные понятия (Классы):**
  - `concept_electronic_component` — класс электронных компонентов (вершины графа).
  - `concept_electrical_connection` — класс электрических соединений (сущность, объединяющая компоненты).
  - `concept_layout_constraint` — класс ограничений на размещение.
  - `concept_optimal_layout` — класс, описывающий результат (оптимальную компоновку).
- **Отношения:**
  - `nrel_connected_to` — ориентированное отношение, показывающее электрическую связь между компонентами.
  - `nrel_in_optimal_layout` — неролевое отношение, связывающее узел результата с выбранными компонентами.
  - `nrel_has_constraint` — отношение наличия ограничения у компонента.
- **Действия (Классы задач):**
  - `action_find_optimal_component_layout` — действие поиска оптимальной компоновки.

2.2 Код основных файлов

```
action_find_optimal_component_layout
<- sc_node_class;
<- concept_class;
=> nrel_main_idtf:
    [действие поиска оптимальной компоновки компонентов] (* <- lang_ru;; *);
    [action to find optimal component layout] (* <- lang_en;; *);;

concept_electrical_connection
<- sc_node_class;
<- concept_class;
=> nrel_main_idtf:
    [электрическое соединение] (* <- lang_ru;; *);
    [electrical connection] (* <- lang_en;; *);;

concept_electronic_component
<- sc_node_class;
<- concept_class;
=> nrel_main_idtf:
    [электронный компонент] (* <- lang_ru;; *);
    [electronic component] (* <- lang_en;; *);;
```

```
concept_layout_constraint
<- sc_node_class;
<- concept_class;
=> nrel_main_idtf:
    [ограничение на размещение] (* <- lang_ru;; *);
    [layout constraint] (* <- lang_en;; *);;

concept_optimal_layout
<- sc_node_class;
<- concept_class;
=> nrel_main_idtf:
    [оптимальная компоновка] (* <- lang_ru;; *);
    [optimal layout] (* <- lang_en;; *);;

nrel_connected_to
<- sc_node_non_role_relation;
<- concept_non_role_relation;
<- concept_binary_relation;
<- concept_oriented_relation;
=> nrel_main_idtf:
    [соединён с*]
    (*
        <- lang_ru;;
    *);
    [connected to*]
    (*
        <- lang_en;;
    *);
=> nrel_first_domain:
    concept_electronic_component;
=> nrel_second_domain:
    concept_electronic_component;;

nrel_has_constraint
<- sc_node_non_role_relation;
<- concept_non_role_relation;
<- concept_binary_relation;
<- concept_oriented_relation;
=> nrel_main_idtf:
    [имеет ограничение*]
    (*
        <- lang_ru;;
    *);
    [has constraint*]
    (*
        <- lang_en;;
    *);
=> nrel_first_domain:
    concept_electronic_component;
=> nrel_second_domain:
    concept_layout_constraint;;

nrel_in_optimal_layout
<- sc_node_non_role_relation;
=> nrel_main_idtf: [входит в оптимальную компоновку*] (* <- lang_ru;; *);
    [in optimal layout*] (* <- lang_en;; *);
=> nrel_first_domain: concept_optimal_layout;
=> nrel_second_domain: concept_electronic_component;;
```

3 РЕАЛИЗАЦИЯ АГЕНТОВ

3.1 Описание логики работы агентов

- FindOptimalCameraPlacementAgent:
  - Агент ищет все компоненты и все соединения.
  - На основе найденных соединений формируется список смежности.
  - Производится упрощенная проверка на планарность графа с использованием формулы Эйлера для плоских графов.
  - Если условие планарности нарушено, вычисляется приблизительное количество необходимых слоев печатной платы.
  - Создается узел concept_optimal_layout, который связывается с компонентами отношением nrel_in_optimal_layout. В текстовый идентификатор результата записывается информация о планарности и количестве слоев.
  - Результат записывается в структуру ответа действия.

3.2 Описание логики работы агентов

Листинг файла pcb_layout_keynodes:

```
#pragma once

#include <sc-memory/sc_keynodes.hpp>

class PcbLayoutKeynodes : public ScKeynodes
{
public:
    static inline ScKeynode const action_find_optimal_component_layout{
        "action_find_optimal_component_layout", ScType::ConstNodeClass};

    static inline ScKeynode const nrel_in_optimal_layout{
        "nrel_in_optimal_layout", ScType::ConstNodeNonRole};

    static inline ScKeynode const nrel_connected_to{
        "nrel_connected_to", ScType::ConstNodeNonRole};

    static inline ScKeynode const concept_electronic_component{
        "concept_electronic_component", ScType::ConstNodeClass};

    static inline ScKeynode const concept_electrical_connection{
        "concept_electrical_connection", ScType::ConstNodeClass};

    static inline ScKeynode const concept_optimal_layout{
        "concept_optimal_layout", ScType::ConstNodeClass};
};
```

3.3 Листинг кода агента

```
#include "find_optimal_layout_agent.hpp"
#include "keynodes/pcb_layout_keynodes.hpp"

#include <sc-memory/sc_memory.hpp>
#include <sc-memory/sc_iterator.hpp>

#include <vector>
#include <algorithm>
#include <unordered_map>

using namespace std;

ScAddr FindOptimalLayoutAgent::GetActionClass() const
{
    return PcbLayoutKeynodes::action_find_optimal_component_layout;
}

ScResult FindOptimalLayoutAgent::DoProgram(ScAction & action)
{
    m_logger.Debug("FindOptimalLayoutAgent started");

    try
    {
        vector<ScAddr> components;
        ScIterator3Ptr itComponents = m_context.CreateIterator3(
            PcbLayoutKeynodes::concept_electronic_component,
            ScType::ConstPermPosArc,
            ScType::ConstNode);

        while (itComponents->Next())
        {
            components.push_back(itComponents->Get(2));
        }

        m_logger.Debug("Found " + to_string(components.size()) + " components");

        vector<pair<ScAddr, ScAddr>> connections;
        ScIterator3Ptr itConnections = m_context.CreateIterator3(
            PcbLayoutKeynodes::concept_electrical_connection,
            ScType::ConstPermPosArc,
            ScType::ConstNode);

        while (itConnections->Next())
        {
            ScAddr connection = itConnections->Get(2);

            vector<ScAddr> connectedComps;
            ScIterator5Ptr itConnected = m_context.CreateIterator5(
                connection,
                ScType::ConstCommonArc,
                ScType::ConstNode,
                ScType::ConstPermPosArc,
                PcbLayoutKeynodes::nrel_connected_to);

            while (itConnected->Next())
            {
                connectedComps.push_back(itConnected->Get(2));
            }
```

```
    for (size_t i = 0; i < connectedComps.size(); ++i)
    {
        for (size_t j = i + 1; j < connectedComps.size(); ++j)
        {
            connections.push_back({connectedComps[i], connectedComps[j]});
        }
    }
}

m_logger.Debug("Found " + to_string(connections.size()) + " connections");

if (components.empty())
{
    m_logger.Warning("No components found");
    return action.FinishSuccessfully();
}

bool isPlanar = true;
int requiredLayers = 1;

if (components.size() >= 3)
{
    int maxEdgesPlanar = 3 * components.size() - 6;

    if (connections.size() > maxEdgesPlanar)
    {
        isPlanar = false;
        requiredLayers = min(4, (int)(connections.size() / maxEdgesPlanar)
        m_logger.Info("Graph is not planar. Estimated layers required: " +
    }
    else
    {
        m_logger.Info("Graph may be planar (E     3V-6 condition satisfied
    }
}

vector<ScAddr> selectedComponents;

if (!isPlanar && !components.empty())
{
    int toSelect = min(8, (int)components.size());
    for (int i = 0; i < toSelect; ++i)
    {
        selectedComponents.push_back(components[i]);
    }
    m_logger.Info("Selected first " + to_string(toSelect) + " components
}
else
{
    selectedComponents = components;
    m_logger.Info("Selected all " + to_string(components.size()) + " compo
}

m_logger.Info("Selected " + to_string(selectedComponents.size()) +
              " components for optimal layout");

ScAddr optimalLayout = m_context.GenerateNode(ScType::ConstNode);

m_context.GenerateConnector(
```

```
        ScType::ConstPermPosArc,
        PcbLayoutKeynodes::concept_optimal_layout,
        optimalLayout
    );

    ScAddr mainIdtfLink = m_context.GenerateLink();
    string idtfText = "Optimal layout of " + to_string(selectedComponents.size
                      " components, planar: " +
                      (isPlanar ? "yes" : "no") +
                      ", layers: " + to_string(requiredLayers);
    m_context.SetLinkContent(mainIdtfLink, idtfText);

    m_context.GenerateConnector(
        ScType::ConstPermPosArc,
        optimalLayout,
        mainIdtfLink
    );
    m_context.GenerateConnector(
        ScType::ConstPermPosArc,
        PcbLayoutKeynodes::nrel_main_idtf,
        m_context.GenerateConnector(
            ScType::ConstCommonArc,
            optimalLayout,
            mainIdtfLink
        )
    );

    for (const auto& component : selectedComponents)
    {
        optimalLayout -> component
        ScAddr arc = m_context.GenerateConnector(
            ScType::ConstCommonArc,
            optimalLayout,
            component
        );

        m_context.GenerateConnector(
            ScType::ConstPermPosArc,
            PcbLayoutKeynodes::nrel_in_optimal_layout,
            arc
        );

        m_logger.Debug("Added component to optimal layout");
    }

    ScStructure result = m_context.GenerateStructure();
    result << optimalLayout;

    for (const auto& component : selectedComponents)
    {
        result << component;
    }

    action.SetResult(result);

    m_logger.Debug("FindOptimalLayoutAgent finished successfully");
    return action.FinishSuccessfully();
}
catch (std::exception const & e)
```

```
    {
        m_logger.Error("FindOptimalLayoutAgent error: " + string(e.what()));
        return action.FinishWithError();
    }
}
```

4 ТЕСТИРОВАНИЕ

4.1 Описание тестов

Для проверки корректности работы были испльзованы Google Test

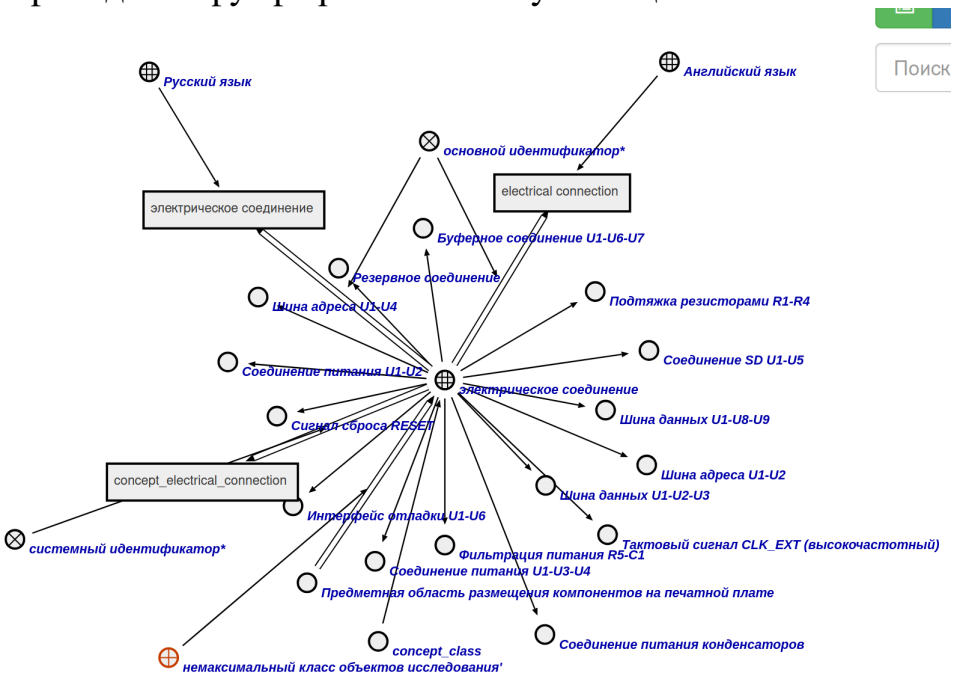
- FindOptimalLayoutAgent_BasicSuccess – проверка успешного создания оптимальной компоновки на базовом наборе данных (3 компонента, 1 соединение). Проверяется наличие результата, узла concept_optimal_layout и связей с компонентами.
- FindOptimalLayoutAgent_NoComponents – тест на обработку пустой базы знаний (отсутствие компонентов). Агент должен завершиться успешно, не создавая результата.
- FindOptimalLayoutAgent_WithPCBSystem – интеграционный тест с имитацией конкретных компонентов (U1, U2, R1). Проверяет способность агента находить произвольные узлы типа concept_electronic_component.

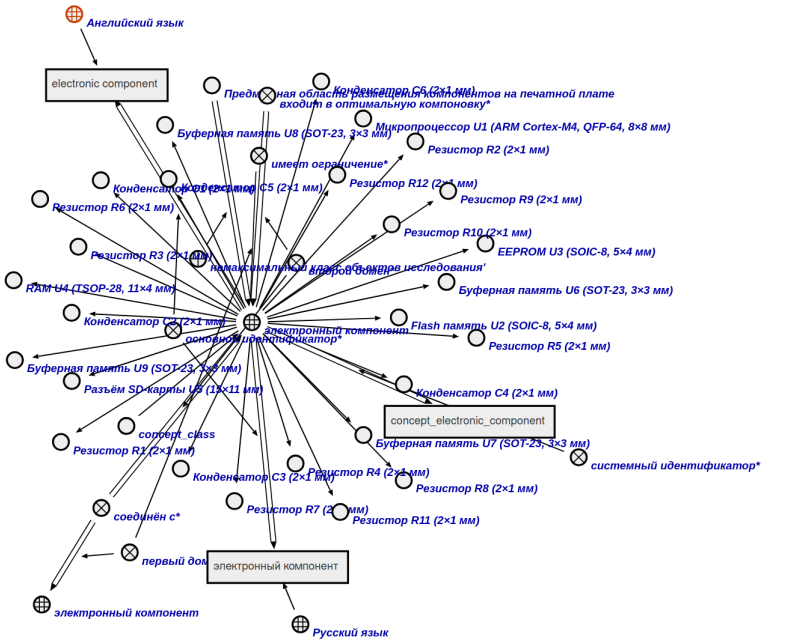
4.2 Успешное выполнение тестов

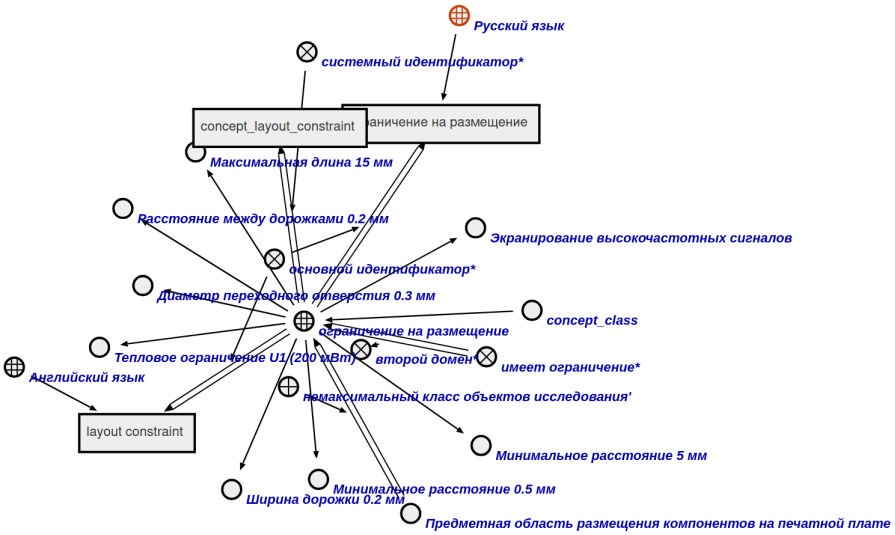
```
[       OK ] AgentTest.FindOptimalLayoutAgent_WithPCBSystem (38 ms)
[----------] 3 tests from AgentTest (135 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (135 ms total)
[  PASSED  ] 3 tests.
```

4.3 Работа системы через web интерфейс

Приведем пару графических визуализаций понятий




Английский язык
electronic component
Предметная область размещения компонентов на печатной плате
входит в оптимальную компоновку*
Конденсатор C6 (2×1 мм)
Буферная память U8 (SOT-23, 3×3 мм)
Микропроцессор U1 (ARM Cortex-M4, QFP-64, 8×8 мм)
имеет ограничение*
Резистор R2 (2×1 мм)
Конденсатор C5 (2×1 мм)
Резистор R12 (2×1 мм)
Резистор R9 (2×1 мм)
Резистор R6 (2×1 мм)
Резистор R10 (2×1 мм)
EEPROM U3 (SOIC-8, 5×4 мм)
Резистор R3 (2×1 мм)
немаксимальный класс объектов исследования'
второй домен
RAM U4 (TSOP-28, 11×4 мм)
Буферная память U6 (SOT-23, 3×3 мм)
Конденсатор C1 (2×1 мм)
электронный компонент
основной идентификатор*
Flash память U2 (SOIC-8, 5×4 мм)
Резистор R5 (2×1 мм)
Буферная память U9 (SOT-23, 3×3 мм)
Разъём SD-карты U5 (11×11 мм)
Конденсатор C4 (2×1 мм)
concept_electronic_component
concept_class
Буферная память U7 (SOT-23, 3×3 мм)
системный идентификатор*
Резистор R1 (2×1 мм)
Конденсатор C3 (2×1 мм)
Резистор R4 (2×1 мм)
Резистор R8 (2×1 мм)
Резистор R7 (2×1 мм)
соединён с*
Резистор R11 (2×1 мм)
первый домен
электронный компонент
электронный компонент
Русский язык


Русский язык
системный идентификатор*
concept_layout_constraint
ограничение на размещение
Максимальная длина 15 мм
Расстояние между дорожками 0.2 мм
Экранирование высокочастотных сигналов
основной идентификатор*
Диаметр переходного отверстия 0.3 мм
concept_class
ограничение на размещение
Тепловое ограничение U1 (200 мВт)
второй домен
имеет ограничение*
Английский язык
немаксимальный класс объектов исследования'
layout constraint
Минимальное расстояние 5 мм
Минимальное расстояние 0.5 мм
Ширина дорожки 0.2 мм
Предметная область размещения компонентов на печатной плате

ЗАКЛЮЧЕНИЕ

В ходе выполнения расчётной работы была разработана интеллектуальная система на базе технологии OSTIS для решения задачи размещения компонентов на печатной плате. Предметная область полностью формализована: введены понятия «электронный компонент», «электрическое соединение», «оптимальная компоновка», а также необходимые отношения. На языке C++ реализован агент FindOptimalLayoutAgent. Агент строит граф соединений на основе данных из sc-памяти, проверяет его планарность используя критерий Эйлера и рассчитывает минимально необходимое количество слоев платы. Результат работы формируется в виде семантической структуры, удобной для дальнейшей обработки или визуализации. Проведено полноценное тестирование: написаны и успешно пройдены модульные тесты с использованием Google Test, покрывающие базовые сценарии, отсутствие данных и работу с множеством компонентов. Решение интегрировано в систему OSTIS и готово к использованию.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Расчётная работа по ППОИС (2025). Методические указания по выполнению расчётной работы [Электронный ресурс]. — 2025. — Дата обращения: 18.05.2025. https://docs.google.com/document/d/1r7dZKbcuxffGYHNGbBZu3UgPRZpjYZYqsxle96-cY90/.

[2] Stroustrup, Bjarne. The C++ Programming Language / Bjarne Stroustrup. — 4th ed. — Addison-Wesley Professional, 2013. — Основной справочник по языку C++ для реализации агентных систем.

[3] Wooldridge, Michael. An Introduction to MultiAgent Systems / Michael Wooldridge. — 2nd ed. — John Wiley & Sons, 2009. — Фундаментальный учебник по принципам агентно-ориентированного программирования.

[4] Boost.Asio c++ library. — 2024. — Библиотека для асинхронного программирования и сетевого взаимодействия агентов. https://www.boost.org/doc/libs/1_84_0/doc/html/boost_asio.html.

[5] The C++ actor framework (CAF). — 2024. — Современный фреймворк для реализации акторной модели на C++. https://www.actor-framework.org/.