



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Компьютерные системы и сети

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА ***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ*** ***БАКАЛАВРА НА ТЕМУ:***

Программная подсистема тестирования знаний
языков описания аппаратуры

Студент

ИУ6-82Б

(Группа)

(Подпись, дата)

С.В. Астахов

(И.О. Фамилия)

Руководитель

(Подпись, дата)

Т.А. Ким

(И.О. Фамилия)

Нормоконтролер

(Подпись, дата)

(И.О. Фамилия)

2023 г.

ЗАДАНИЕ

КАЛЕНДАРНЫЙ ПЛАН

АННОТАЦИЯ

В настоящее время существует огромное количество образовательных ресурсов, посвященных тематике информационных технологий. Несмотря на это, на данный момент в открытом доступе наблюдается дефицит ресурсов, посвященных изучению языков описания аппаратуры. Среди существующих образовательных платформ есть лишь несколько таких, на которых возможно настроить автоматическую проверку заданий на написание исходного кода на языке Verilog (или каком-либо другом языке описания аппаратуры) непосредственно в рамках веб-приложения. При этом процесс настройки весьма сложен, поэтому авторы курсов редко используют описанную возможность.

Настоящая квалификационная работа посвящена разработке программной подсистемы тестирования знаний языков описания аппаратуры, которая предоставляет возможности по управлению учебными материалами и автоматической проверки заданий (в том числе, заданий на описание аппаратных устройств на языке Verilog) в рамках информационной системы образовательного портала.

В исследовательской части работы представлены результаты анализа существующих систем тестирования знаний с целью уточнения функциональных требований и определения вариантов использования разработанной подсистемы.

В конструкторской части работы спроектирована программная подсистема тестирования знаний языков описания аппаратуры, позволяющая управлять учебными материалами и заданиями, проводить автоматическую проверку решений пользователей (учащихся). Подсистема предназначена для интеграции в информационную систему образовательного портала.

В технологической части была разработана технология тестирования разработанной подсистемы, проведено ее функциональное и нагрузочное тестирование.

Процесс настройки и развертывания подсистемы описан в руководстве системного программиста.

ABSTRACT

Currently, there are a huge number of educational resources dedicated to the topic of information technology. Despite this, at the moment there is a shortage of resources in the public domain devoted to the study of the languages of the description of the apprature. Among the existing educational platforms, there are only a few on which it is possible to set up automatic verification of tasks for writing source code in the Verilog language (or some other hardware description language) directly within the web application. At the same time, the setup process is very complicated, so the authors of the courses rarely use the described feature.

This qualification work is devoted to the development of a software subsystem for testing knowledge of hardware description languages, which provides opportunities for managing educational materials and automatically checking tasks (including tasks for describing hardware devices in Verilog) within the educational portal information system.

In the research part of the work, an analysis of existing knowledge testing systems was carried out in order to clarify the functional requirements and determine the use cases of the developed subsystem.

In the design part of the work, a software subsystem for testing knowledge of hardware description languages has been designed, which allows you to manage educational materials and tasks, and automatically check the decisions of users (students). The subsystem is designed for integration into the information system of the educational portal.

In the technological part, the technology for testing the developed subsystem was developed, its functional and load testing was carried out.

The process of configuring and deploying the subsystem is described in the system programmer's manual.

РЕФЕРАТ

Расчетно-пояснительная записка 67 стр., 30 рис., 12 табл., 15 источников, 6 приложений.

ТЕСТИРОВАНИЕ ЗНАНИЙ, ЯЗЫК ОПИСАНИЯ АППАРАТУРЫ, HDL, VERILOG, СИСТЕМА ДИСТАНЦИОННОГО ОБУЧЕНИЯ, ОБРАЗОВАТЕЛЬНЫЙ ПОРТАЛ.

Объектом разработки является программная подсистема тестирования знаний языков описания аппаратуры.

Цель работы — разработать программную подсистему тестирования знаний языков описания аппаратуры, реализующую следующие функции:

- добавление, удаление, редактирование образовательных материалов и заданий (для администратора);
- проверка правильности решения заданий пользователями (в т.ч. заданий на написание программного кода);
- анализ ошибок в пользовательских решениях;
- занесение результатов решения в базу данных;
- анализ статистики решения заданий.

В результате разработки была спроектирована и реализована программная подсистема тестирования знаний языков описания аппаратуры, разработана технология ее тестирования и проведено ее функциональное и нагрузочное тестирование.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ.....	9
ВВЕДЕНИЕ.....	10
1 Анализ систем тестирования знаний языков программирования.....	12
1.1 Классификация методов тестирования знаний.....	12
1.2 Методы тестирования знаний	13
1.2.1 Тестирование с ответом в закрытой форме	13
1.2.2 Тестирование с коротким ответом и ответом в форме эссе.....	15
1.2.3 Проверка программ по референсным значениям.....	17
1.2.4 Автоматизированное тестирование программ на проверяющей стороне.....	19
1.3 Функциональные требования и диаграмма вариантов использования подсистемы.....	20
1.4 Выводы.....	23
2 Проектирование программной подсистемы тестирования знаний языков описания аппаратуры.....	24
2.1 Проектирование архитектуры и бизнес-логики	24
2.2 Проектирование базы данных и структур данных.....	27
2.2.1 Разработка даталогической схемы БД	27
2.2.2 Описание структур данных заданий и ответов.....	29
2.3 Проектирование микросервисов.....	31
2.3.1 Микросервис взаимодействия с БД.....	31
2.3.2 Микросервис синтеза устройств (синтезатор).....	36
2.3.3 Микросервисы для работы с временными диаграммами.....	37
2.3.4 Микросервис анализа статистики.....	42
2.3.5 Микросервис анализа решений (анализатор).....	43
2.4 Выводы.....	44

3 Разработка технологии тестирования.....	45
3.1 Выбор подходов и методов тестирования.....	45
3.2 Разработка плана автономного тестирования.....	46
3.4 Выбор языка программирования и библиотек для функционального тестирования.....	53
3.5 Реализация и проведение функциональных тестов	54
3.6 Нагрузочное тестирование	59
3.7 Выводы.....	64
ЗАКЛЮЧЕНИЕ.....	65
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	66
Приложение А. Техническое задание.....	68
Приложение Б. Руководство системного программиста.....	69
Приложение В. Графический материал.....	70
Приложение Г. Исходный код микросервиса анализа решений.....	71
Приложение Д. Исходные коды на языке Verilog.....	78
Приложение Е. Временная диаграмма в формате VCD.....	80

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке к ВКРБ применяются следующие обозначения и сокращения:

БД — база данных;

ПЛИС — программируемая логическая интегральная схема;

ПО — программное обеспечение;

СДО — система дистанционного обучения;

СУБД — система управления базами данных;

ТЗ — техническое задание;

API (Application Programming Interface) — интерфейс прикладного программирования;

C4 (Context Container Component Code) — контекст-контейнер-компонент-код;

CRUD (Create Read Update Delete) — создание-чтение-изменение-удаление;

GPL (General Public License) — универсальная общественная лицензия;

HTTP (HyperText Transfer Protocol) — протокол передачи гипертекста;

IDEF-0 (Integration Definition for Function Modeling) — Определение интеграции для функционального моделирования;

JSON (JavaScript Object Notation) — нотация объекта JavaScript;

VCD (Value Change Dump) — дамп изменения значения.

ВВЕДЕНИЕ

Несмотря на наличие большого числа теоретических материалов, посвященных языкам описания аппаратуры, в настоящее время наблюдается дефицит и низкое качество организации ресурсов, ориентированных на их практическое освоение. Абсолютное большинство таких ресурсов (marsohod.org, portal-ed.ru, asic-world.com) предоставляет лишь теоретические данные и набор практических упражнений, которые пользователю предлагается выполнить в стороннем программном обеспечении.

Такой подход может быть довольно сложен для новичка в силу описанных ниже проблем.

Первая проблема — установка стороннего программного обеспечения. Наиболее часто используемые для работы с языками описания аппаратуры среды: Quartus и Xilinx. Обе они требуют большого объема как постоянной, так и оперативной памяти. Кроме того, для приобретения начальных навыков функциональность этих сред избыточна, так как значительная ее часть ориентирована на адаптацию проекта под конкретную аппаратную базу для дальнейшей прошивки в ПЛИС. Избыточная функциональность (с точки зрения рассматриваемой задачи) требует дополнительных вычислительных ресурсов и усложняет работу пользователя с этими средами.

Альтернативой Xilinx и Quartus являются такие инструменты, как Icarus Verilog. Это легковесная среда симуляции и синтеза устройств, описанных на языке Verilog. Взаимодействие с пользователем осуществляется через консольный интерфейс, результаты симуляции записываются в VCD-файл и затем отображаются графически через такие утилиты, как GTKWave. Основным недостатком в этом случае являются непривычный для новичка консольный интерфейс.

Вторая проблема — отсутствие внешнего контроля и системы оценивания. Безусловно, большинство людей в состоянии объективно оценить правильность функционирования описанного ими устройства по временным

диаграммам, полученным в результате запуска Testbench-файлов, прикрепленных к заданию. Однако, обучение на основе только таких заданий не позволяет закрепить теоретические знания, которые можно было бы проверить, например, тестовыми заданиями. Кроме того, такая система усложняет контроль человека за освоением курса в целом, утрачивается ощущение объективности оценки собственного прогресса, ухудшается качество обучения [1].

Стоит отметить, что полноценное освоение языков описания аппаратуры в принципе затруднительно без знаний в области цифровой схемотехники, архитектуры ЭВМ и т.п. Однако, цель образовательных платформ, посвященных этой тематике состоит прежде всего именно в формировании базовых знаний и навыков работы с языками описания аппаратуры для людей, интересующихся ими, например, в качестве хобби или с целью продолжить обучение в университете и т.п.

1 Анализ систем тестирования знаний языков программирования

Работа, проведенная в рамках исследовательской части позволяет подойти к решению описанных во введении проблем посредством формирования функциональных требований и составления диаграммы вариантов использования подсистемы тестирования знаний языков описания аппаратуры, полученных на основе анализа имеющихся систем тестирования знаний языков программирования и методов тестирования знаний, используемых в этих системах.

1.1 Классификация методов тестирования знаний

Перед функциональным моделированием различных методов тестирования знаний, необходимо ввести их классификацию.

В качестве основы была взята подобная классификация для системы дистанционного обучения (далее — СДО) Moodle [2]. Она была дополнена с учетом функциональных особенностей таких СДО, как Huawei University, Coursera, Stepik, Ethernaut, а также хакатона Paradigm CTF [3]. Сформированная классификация приведена в таблице 1.

Таблица 1 — Классификация методов тестирования знаний

№	Тип	Подтип
1	Тестирование с ответом в закрытой форме	1.1 Выбор одного ответа 1.2 Выбор множественных ответов 1.3 Сопоставление
2	Тестирование с коротким ответом	2.1 С автоматизированной проверкой 2.2 С проверкой преподавателем 2.3 С перекрестной проверкой
3	Тестирование с ответом в форме эссе	3.1 С проверкой преподавателем 3.2 С перекрестной проверкой

Продолжение таблицы 1

4	Тестирование на написание исходного кода	<p>4.1 С проверкой по референсным значениям</p> <p>4.2 Автоматизированное тестирование на проверяющей стороне</p> <p>4.3 Другие</p>
---	--	---

1.2 Методы тестирования знаний

1.2.1 Тестирование с ответом в закрытой форме

Тестирование с ответом в закрытой форме применяется практически во всех системах тестирования знаний. Соответствующая IDEF0-модель представлена на рисунке 1.

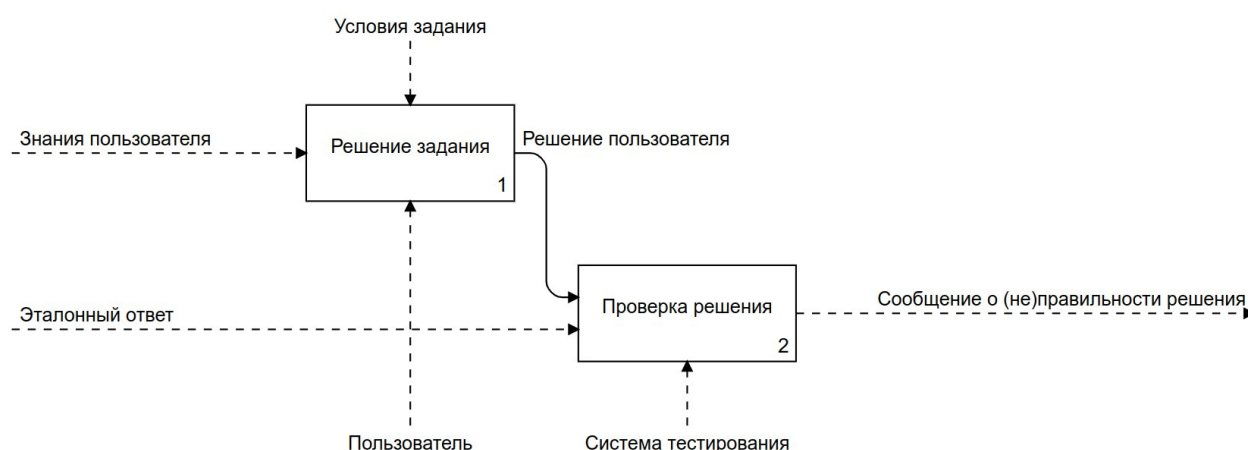


Рисунок 1 — Функциональная модель тестирования с ответом в закрытой форме

Основным недостатком такой реализации тестирования с ответом в закрытой форме является невозможность получить содержательную обратную связь в случае неверного решения. Возможные формы обратной связи для различных подтипов заданий с закрытым ответом показаны в таблице 2.

Таблица 2 — Формы обратной связи для тестирования с ответом в закрытой форме

Подтип тестирования	Форма обратной связи
Выбор одного ответа	Пояснение причин некорректности ответа
Выбор множественных ответов	Сообщение о выборе избыточного/недостаточного числа вариантов
Сопоставление	Сообщение о количестве неправильно выбранных пар
	Подсветка некорректно выбранных пар

В случае заданий на выбор множественных ответов сообщение о выборе избыточного/недостаточного числа вариантов неинформативно и не позволяет пользователю повторно проанализировать задание с его учетом. При этом такой вид обратной связи позволяет пользователю сократить число вариантов для перебора ответов при повторном решении задания. По этим причинам использование обратной связи в заданиях этого подтипа не всегда желательно.

В случае заданий на сопоставление сообщение о количестве неправильно выбранных пар менее информативно, но не сокращает число вариантов перебора ответа. Подсветка некорректно выбранных пар содержит полезную для повторного анализа задания информацию, но сокращает число вариантов перебора ответа.

Еще одной проблемой тестирования с ответом в закрытой форме, уже затронутой выше, является проблема перебора ответов. Данная проблема не возникает в случае проведения контрольных мероприятий, где количество попыток прохождения тестирования ограничено. Однако, в случае открытых онлайн-курсов, число попыток прохождения тестирования, как правило, не ограничивается. В таком случае одним из решений проблемы является ограничение времени до возможности повторно пройти тестирование.

Функциональная модель прохождения тестирования с ответом в закрытой форме с учетом предложенных улучшений представлена на рисунке 2.

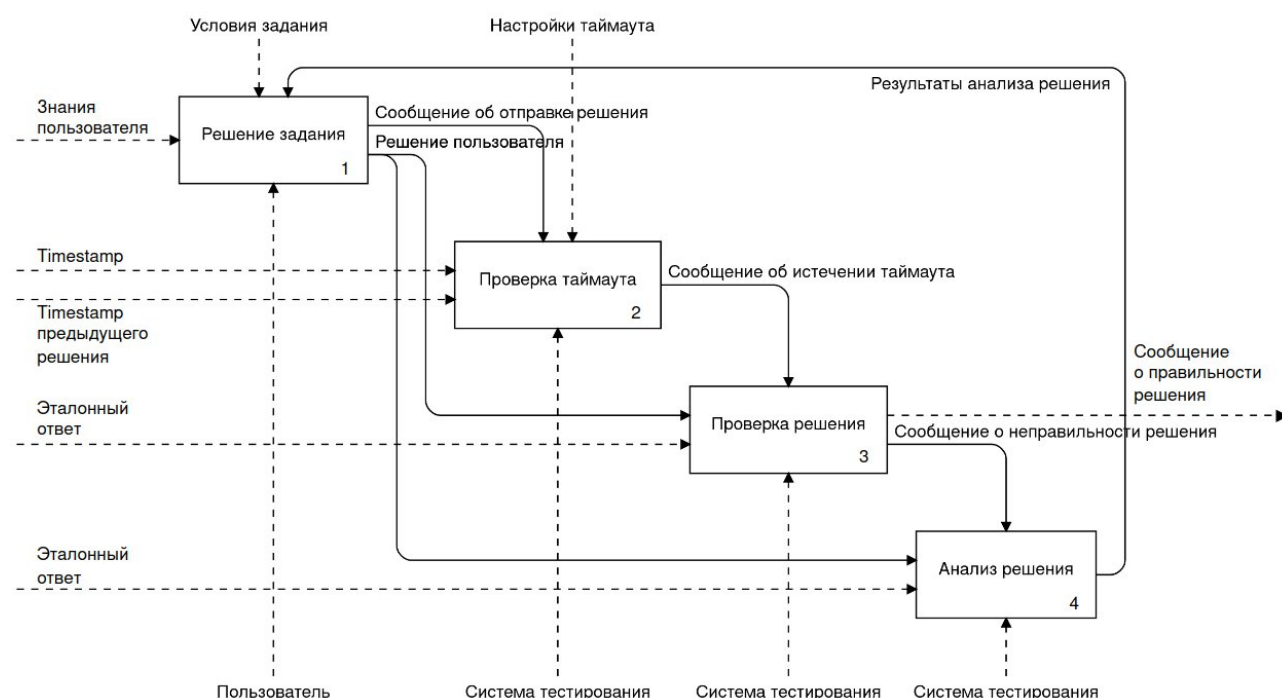


Рисунок 2 — Усовершенствованная функциональная модель тестирования

1.2.2 Тестирование с коротким ответом и ответом в форме эссе

Тестирование с коротким ответом может быть проверено автоматически, преподавателем или участниками тестирования перекрестно. Автоматизированная проверка зачастую не учитывает все возможные формы слова, синонимы, грамматические ошибки в ответе и т.п. Однако, она не требует вовлечения преподавателя или перекрестной оценки, которая может быть субъективной в силу тех или иных причин. Поэтому, в случае открытых онлайн-курсов, ориентированных на большое число участников, как правило используется автоматизированная проверка таких вопросов. В случае же каких-либо контрольных мероприятий в рамках, например, СДО университета, рекомендуется использовать проверку ответов преподавателем, чтобы избавиться от технических ошибок при проверке задания в автоматизированном режиме.

Кроме того, стоит отметить, что данный тип тестирования, используемый, например, для проверки решения задач по математике, не

позволяет предоставить пользователю информативную обратную связь о его ошибках.

Тестирование в форме эссе применяется как правило для контрольных мероприятий в рамках СДО университета и т.п. В этом случае задание проверяется преподавателем. В открытых онлайн-курсах такие задания как правило не применяются, так как преподавательского ресурса недостаточно для проверки заданий всех пользователей, число которых может быть очень велико. В редких случаях, когда использование такой формы тестирования все же необходимо (например, при отсутствии технической возможности проверить задачу на программирование), прибегают к системе перекрестной проверки. В таком случае требования к ответу стараются максимально формализовать, чтобы пользователи могли более объективно оценить друг друга. При перекрестном тестировании в качестве итоговой оценки, как правило, выставляется среднее или медианное значение результатов нескольких проверок [4].

Пример интерфейса проверки задания с перекрестным оцениванием на платформе Stepik приведен на рисунке 3.

The screenshot displays the 'Рецензия' (Review) interface on the Stepik platform. It is divided into two main sections: a review form on the left and a list of criteria on the right.

Review Form (Left):

- Шаг:** [PyTest — параметризация, конфигурирование, плагины Шаг 9](#)
- Решение:** #787527998, 23 октября 2022 г., 22:04, верно
- Условие:** [Text area] Развернуть ▾
- Решение #787527998**
- Не забудьте отправить ваше решение на рецензирование.
-

Criteria (Right):

- Критерий 1**
Тест в репозитории можно запустить командой `pytest --language=es`, тест успешно проходит.
Оценка *
☐ 0 ☒ 1

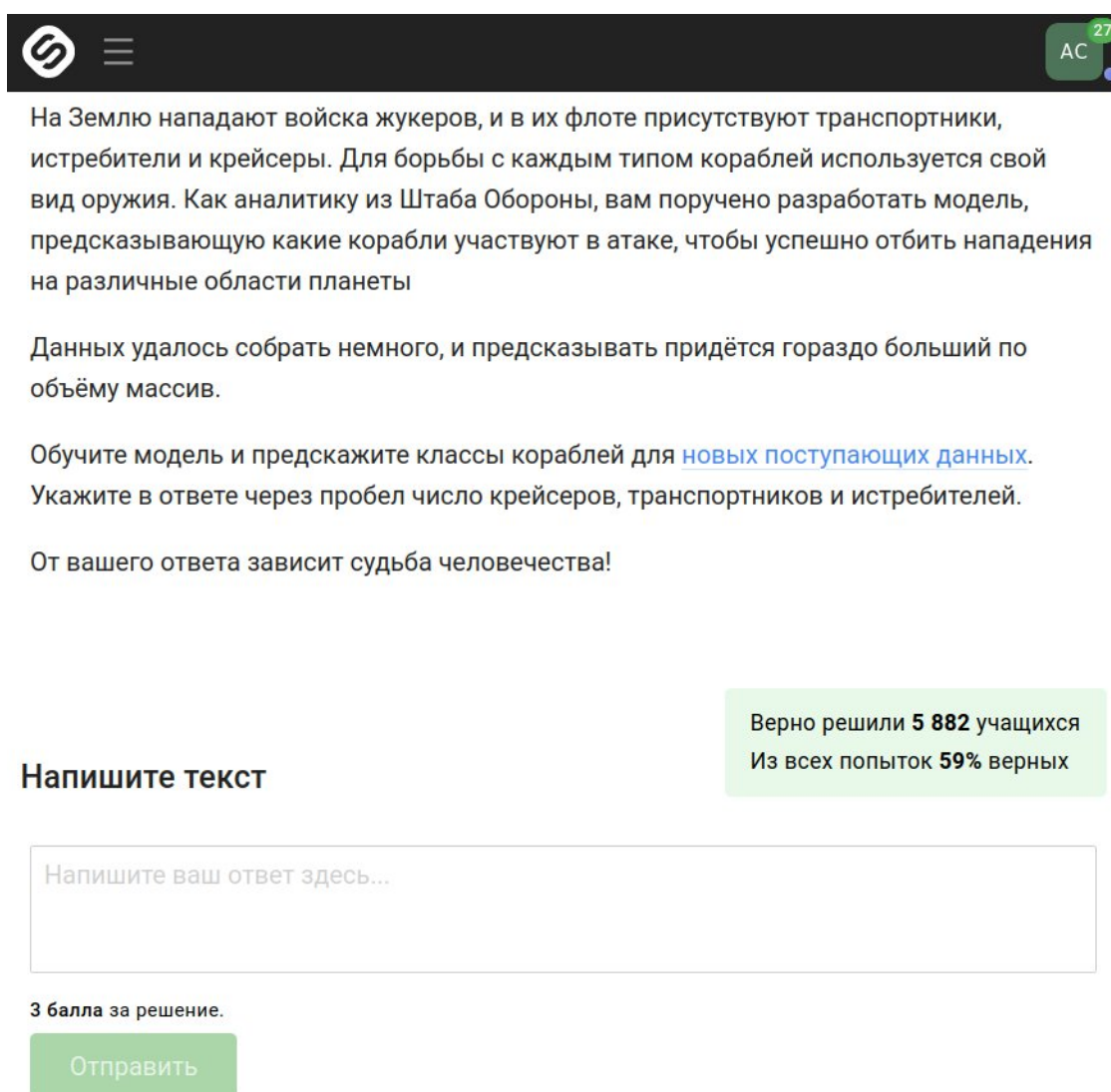
* Обязательное поле
- Критерий 2**
Проверка работоспособности кода для разных языков. Добавьте в файл с тестом команду `time.sleep(30)` сразу после открытия ссылки. Запустите тест с параметром `--language=fr` и визуально проверьте, что фраза на кнопке добавления в корзину выглядит так: **"Ajouter au panier"**.
Не снижайте баллы за отсутствие `time.sleep(30)`, предполагается, что вы его добавите самостоятельно.

Рисунок 3 — Интерфейс проверки задания с перекрестным оцениванием на платформе Stepik

1.2.3 Проверка программ по референсным значениям

Зачастую, так как разработчики онлайн-портала не обладают достаточными ресурсами для создания подсистемы автоматизированного тестирования пользовательских программ, либо сама архитектура проверяемой программы не позволяет протестировать ее автоматически по техническим причинам (например, сама программа пользователя связана с тематикой автоматизированного тестирования, программа связана с машинным обучением и потребляет много вычислительных ресурсов и т.д.)

Пример таких заданий приведены на рисунках 4 и 5.



The screenshot shows a task interface on a dark-themed portal. At the top, there is a logo on the left and a green 'AC' badge with the number '27' on the right. The main text area contains a task description in Russian about predicting ship classes based on historical data. Below the text, there is a green box with statistics: 'Верно решили 5 882 учащихся' and 'Из всех попыток 59% верных'. The task prompt 'Напишите текст' is followed by a large text input field with a placeholder 'Напишите ваш ответ здесь...'. Below the input field, it says '3 балла за решение.' and there is a green 'Отправить' button.

На Землю нападают войска жукеров, и в их флоте присутствуют транспортники, истребители и крейсеры. Для борьбы с каждым типом кораблей используется свой вид оружия. Как аналитику из Штаба Оборона, вам поручено разработать модель, предсказывающую какие корабли участвуют в атаке, чтобы успешно отбить нападения на различные области планеты

Данных удалось собрать немного, и предсказывать придётся гораздо больший по объёму массив.

Обучите модель и предскажите классы кораблей для [новых поступающих данных](#). Укажите в ответе через пробел число крейсеров, транспортников и истребителей.

От вашего ответа зависит судьба человечества!

Верно решили **5 882** учащихся
Из всех попыток **59%** верных

Напишите текст

Напишите ваш ответ здесь...

3 балла за решение.

Отправить

Рисунок 4 — Проверка задания на машинное обучение по референсным значениям

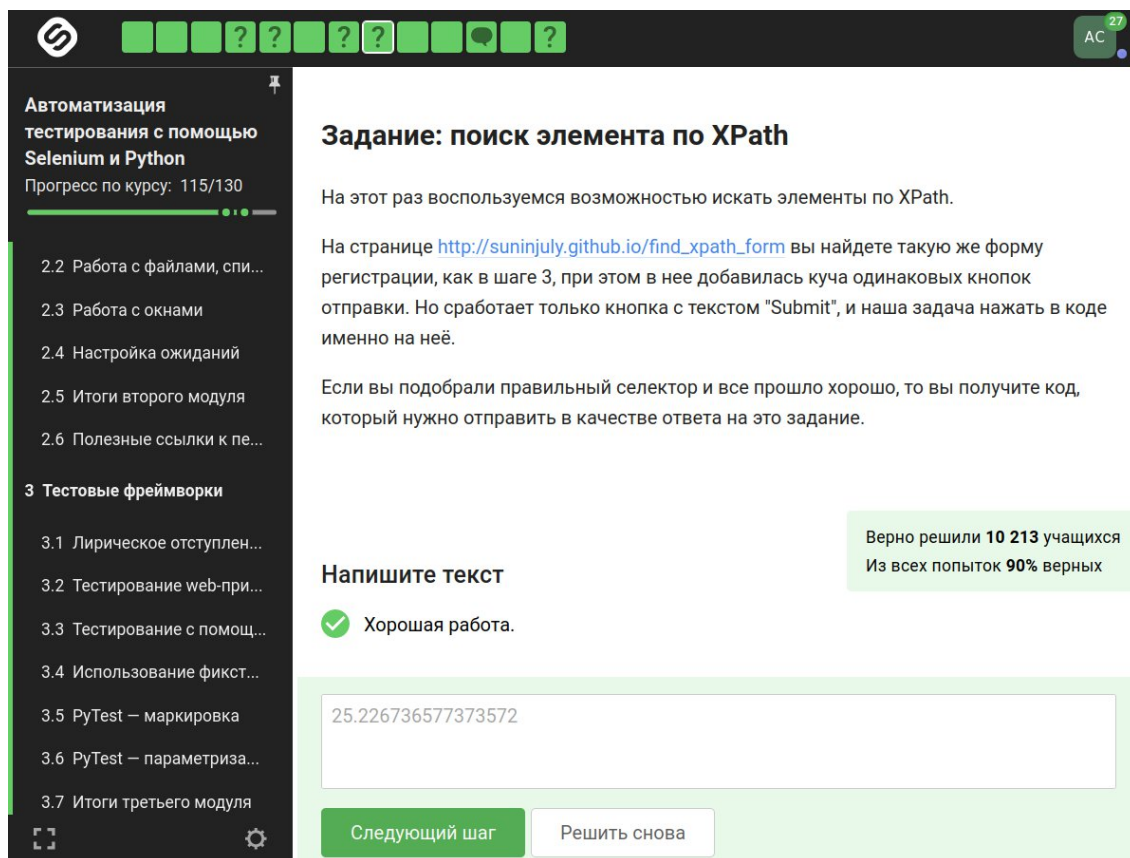


Рисунок 5 — Проверка задания на автоматизированное тестирование по референсным значениям

Функциональная модель тестирования на написание программы с проверкой по референсным значениям приведена на рисунке 6.

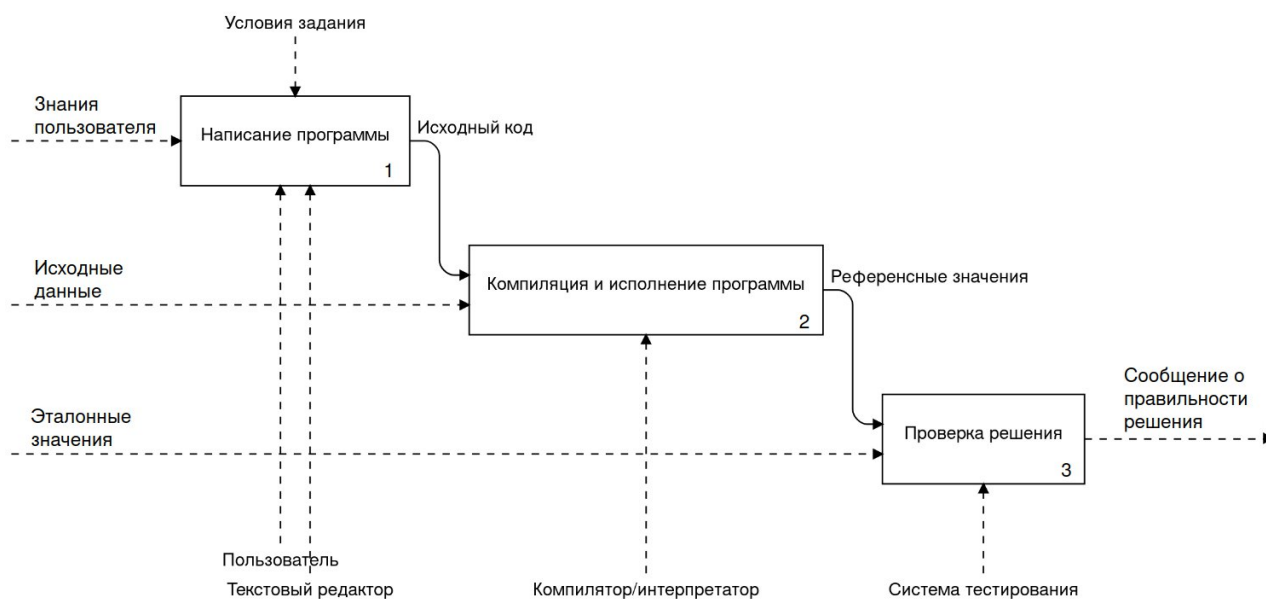


Рисунок 6 — Функциональная модель тестирования на написание программы с проверкой по референсным значениям

При анализе представленной функциональной модели становится очевиден ряд недостатков такого типа тестирования.

Первым недостатком является отсутствие информативной обратной связи, которое затрудняет пользователю поиск семантических ошибок в логике программы.

Вторым недостатком является необходимость установки дополнительного программного обеспечения (текстового редактора и компилятора, либо среды разработки) со стороны пользователя. Это не только повышает входной порог, но и лишает пользователя возможности проходить обучение и тестирование без своего компьютера (например, с мобильного устройства в общественном транспорте, во время командировки или путешествия).

Третьим недостатком является необходимость составления таких заданий и подбор таких входных данных, результаты которых достаточно сложно или невозможно рассчитать без написания требуемой программы. Зачастую, этот процесс может быть затруднительным и в итоге потребует от пользователя написания более сложной программы, чем в случае если бы задание было нацелено исключительно на формирование и проверку целевого навыка.

1.2.4 Автоматизированное тестирование программ на проверяющей стороне

Наиболее каноничным способом проверки заданий по программированию является автоматизированное тестирование на проверяющей стороне. Функциональная модель тестирования написание программы с автоматизированной проверкой показана на рисунке 7.

Положительными сторонами такого подхода к проверке заданий на программирования являются [5]:

- объективность оценки (оцениваемые программы проходят через одинаковый набор тестов или эквивалентные между собой наборы тестов);

- скорость оценки;
- наглядность оценки (хотя процесс тестирования проходит в режиме черного ящика, результат тестирования, при должной подготовки системы автоматизированного и самих тестов, нагляден и способен в некоторой степени оповестить об ошибках в тестируемых приложениях);
- возможность применения для большого числа пользователей.

Основным недостатком такого подхода является сложность его реализации.

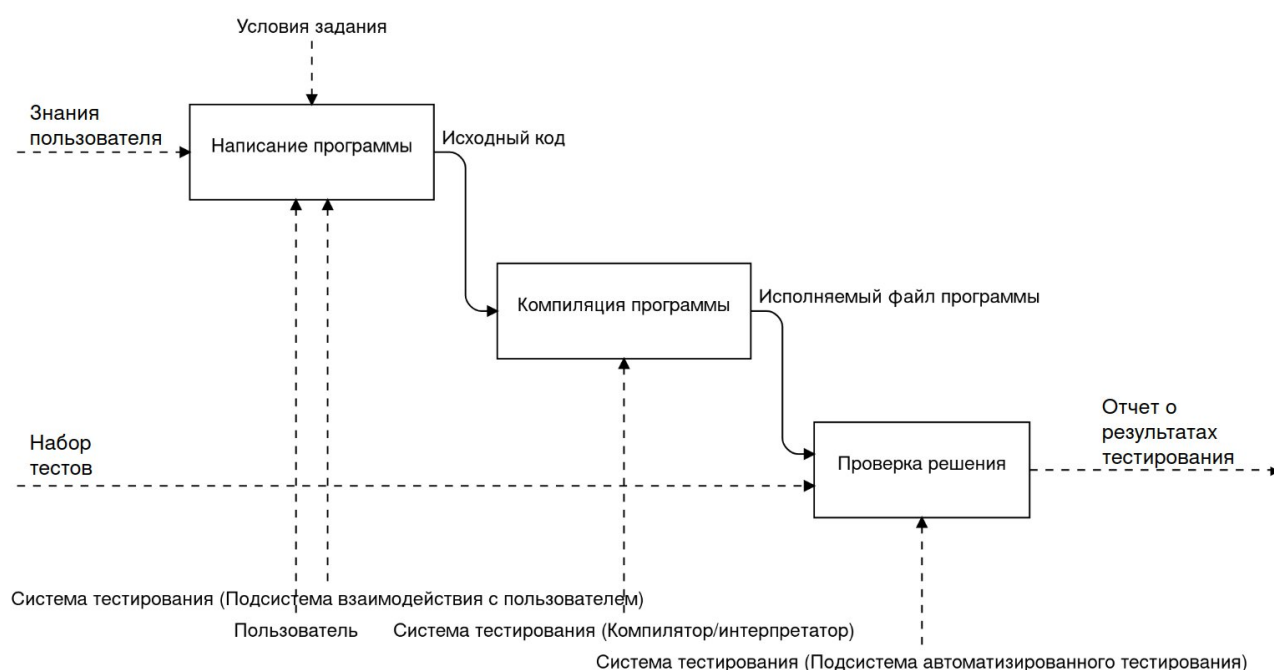


Рисунок 7 — Функциональная модель тестирования на написание программы с автоматизированной проверкой

1.3 Функциональные требования и диаграмма вариантов использования подсистемы

Из проанализированных методов тестирования и оценивания знаний знаний были выделены наиболее подходящие для использования в подсистеме тестирования знаний языков описания аппаратуры (таблица 3).

Таблица 3 — Методы тестирования знаний в проектируемой системе

№	Тип	Подтип	Вид обратной связи
1	Тестирование с ответом в закрытой форме	Выбор одного ответа	Текстовое пояснение ошибки
		Выбор нескольких ответов	Информации о наличии ложноположительных (ложноотрицательных) ответов
2	Задание на написание исходного кода	Автоматизированное тестирование на проверяющей стороне	Информация о несоответствующих сигналах

На основе результатов проведенного анализа было заключено, что проектируемая подсистема должна выполнять следующие функции:

- изменение заданий модератором;
- отображение персональной статистики учащегося;
- обработка статистики решения заданий;
- формирование рейтингового списка учащихся;
- автоматизированная проверка тестов с закрытым ответом, кратким ответом и ответом в виде исходного кода;
- формирование информативной обратной связи в случае неверного решения задания учащимся;
- формирование временных диаграмм работы устройств.

На основе результатов проведенного анализа и сформулированных функциональных требований была разработана диаграмма вариантов использования подсистемы тестирования знаний языков описания аппаратуры, представленная на рисунке 8 [6].

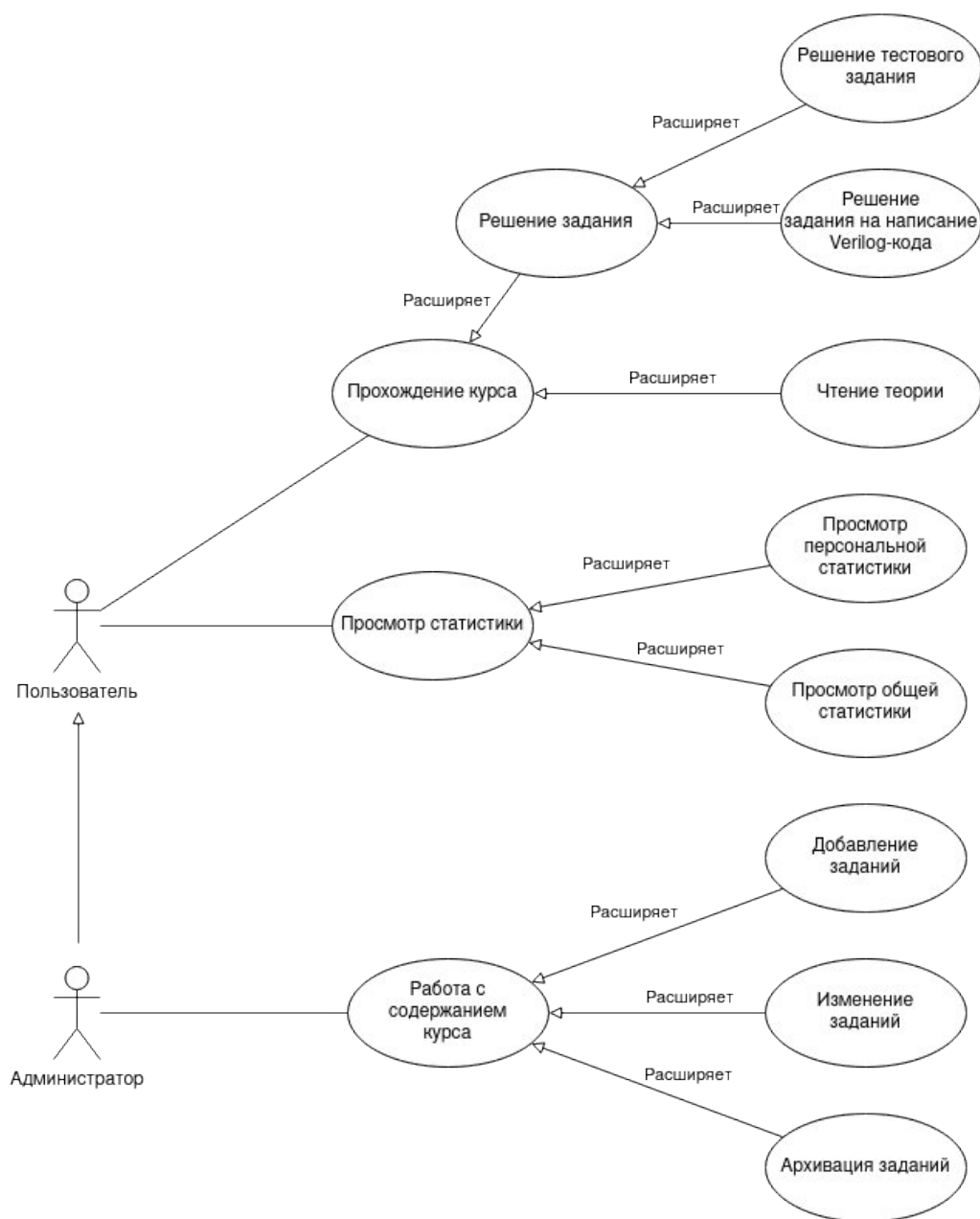


Рисунок 8 — Диаграмма вариантов использования подсистемы тестирования знаний языков описания аппаратуры

Полученная диаграмма вариантов использования предусматривает 3 базовых варианта использования: прохождение курса, просмотр статистики, работа с содержанием курса.

Работа с учетными данными пользователей, отрисовка пользовательского интерфейса и другие стандартные функции реализуются вне рамок проектируемой подсистемы.

1.4 Выводы

В исследовательской части проведен анализ функциональных возможностей существующих платформ обучения языкам программирования, получены и доработаны их функциональные модели, на их основе сформулированы функциональные требования и диаграмма вариантов использования подсистемы тестирования знаний языков описания аппаратуры.

Основными отличиями такой подсистемы от аналогов являются:

- реализация модуля автоматизированного тестирования на основе временных диаграмм, описываемых VCD-файлами;
- учет статистики прохождения заданий;
- информативная обратная связь об ошибках учащегося;
- формирование рейтинга учащихся.

2 Проектирование программной подсистемы тестирования знаний языков описания аппаратуры

2.1 Проектирование архитектуры и бизнес-логики

Разработанная подсистема используется веб-приложением образовательного портала для управления учебными материалами, проверки пользовательских ответов на задания и работы со статистикой решения заданий.

Поскольку информация о пользователях используется как в разработанной подсистеме, так и в других компонентах программного обеспечения образовательного портала, БД используется совместно.

Обобщенная архитектура информационной системы показана с помощью контекст-диаграммы в нотации С4 на рисунке 9 [7].

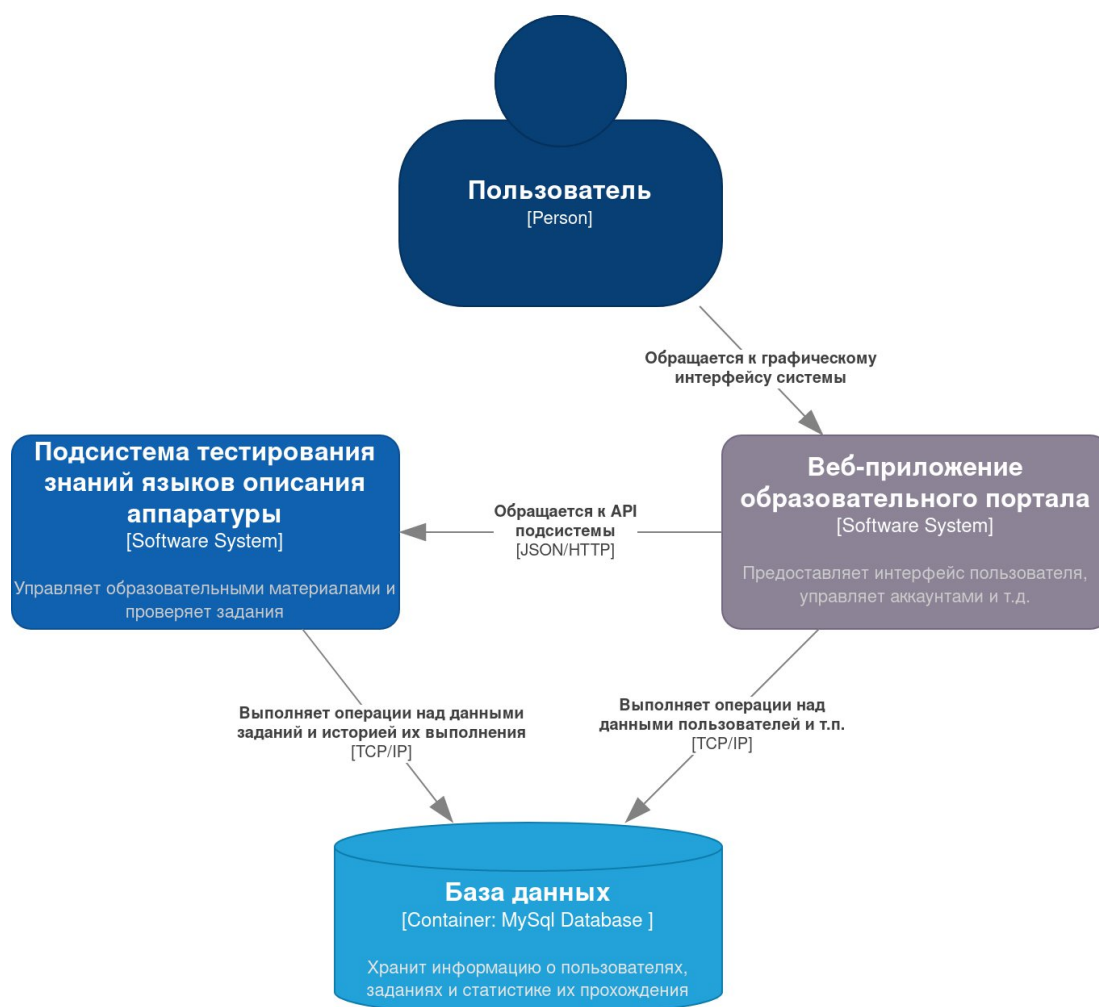


Рисунок 9 — Обобщенная архитектура информационной системы

Так как разработанная подсистема является весьма сложной, выполняемые в ней операции разнородны, могут потребовать использования различных языков и библиотек, а также некоторые из них могут занимать значительное время, при разработке было решено использовать микросервисную архитектуру.

Микросервисный подход обладает следующими преимуществами:

- позволяет использовать различные языки программирования для реализации различных компонентов;
- упрощает тестирование;
- позволяет использовать горизонтальное масштабирование для различных компонентов в зависимости от нагрузки на них.

На основе функциональных требований, предъявляемых к разработанной подсистеме и представленной в главе 1 диаграммы вариантов использования была разработана структура компонентов, отвечающих за их реализацию:

- микросервис взаимодействия с БД — реализует CRUD-операции над данными в БД;
- микросервис анализа решений (анализатор) — выполняет проверку и анализ пользовательских решений;
- микросервис синтеза устройств (синтезатор) — выполняет синтез устройств из Verilog-кода и симулирует их работу;
- микросервис разбора временных диаграмм, микросервис генерации временных диаграмм wavedrom — преобразуют временные диаграммы в удобные для хранения и обработки форматы;
- микросервис анализа статистики;
- основной микросервис — реализует бизнес-логику подсистемы, связывает остальные микросервисы.

Детализированная архитектура разработанной подсистемы показана на контейнер-диаграмме (нотация C4) на рисунке 10.

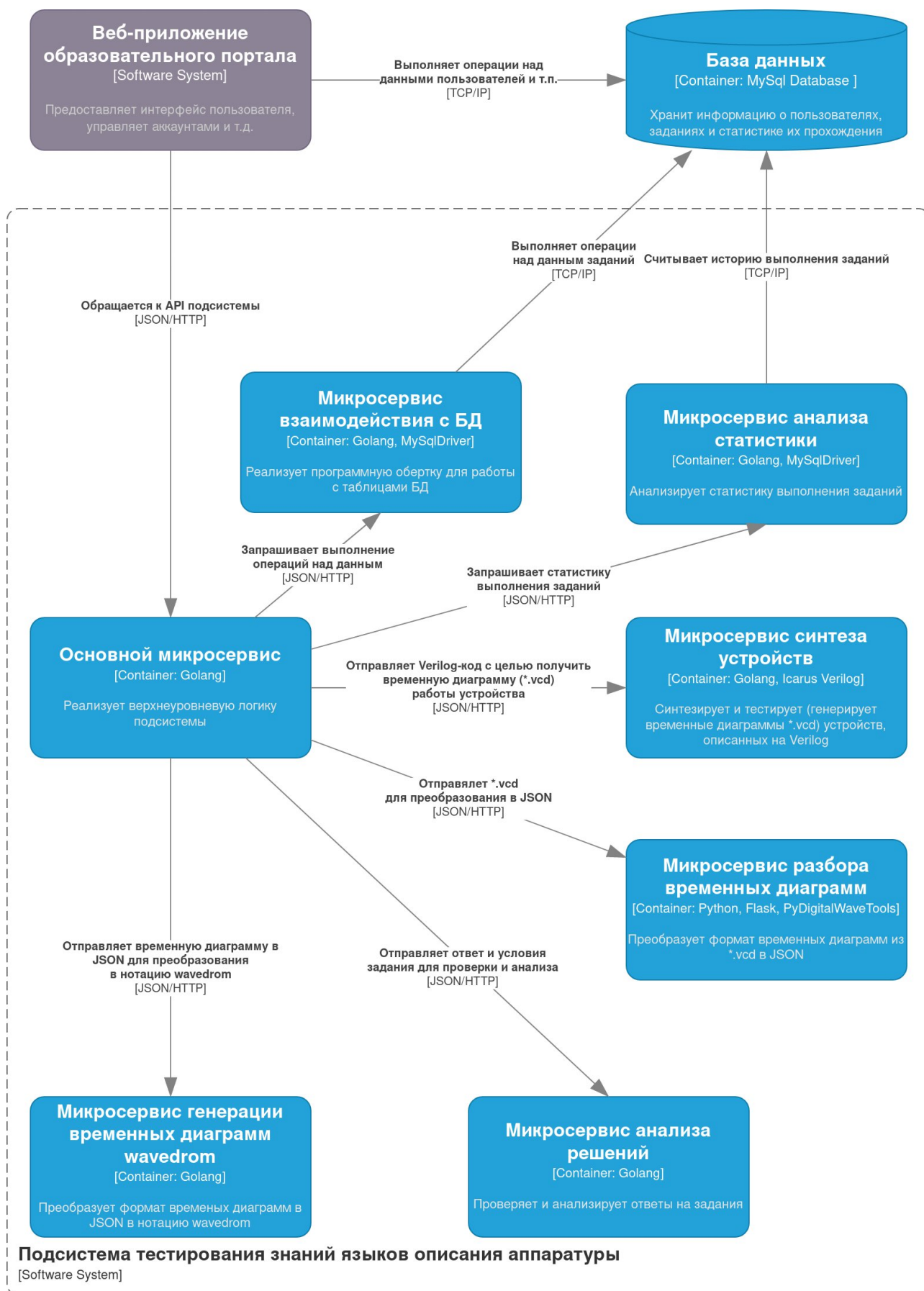


Рисунок 10 — Детализированная архитектура разработанной подсистемы

2.2 Проектирование базы данных и структур данных

2.2.1 Разработка даталогической схемы БД

В результате анализа предметной области удалось выделить описанные ниже сущности.

Сущность «Задание» — содержит информацию о порядковом номере задания, его условиях, правильном ответе, цене в баллах и т.п.;

Сущность «Пользователь» — позволяет идентифицировать пользователя по ID, узнать, обладает ли пользователь правами администратора и узнать его псевдоним (т.н. «никнейм»). Кроме того, эта сущность может нести в себе дополнительную информацию, необходимую веб-приложению образовательного портала.

Сущность «Попытка решения» — содержит информацию, об успешности и времени каждой попытки решения задания каким-либо пользователем.

Для реализации базы данных была выбрана реляционная СУБД MySQL, для ускорения работы SQL-запросов, анализирующих статистику прохождения заданий или выдающих другую агрегированную информацию по курсу, было решено разделить сущность «Задание» на «Брифинг задания» и «Данные задания», а так же выделить отдельную сущность «Тип задания».

Для кратких текстовых полей, таких, как «Название задания» используется тип var, а для длинных — TEXT. Логические значения сохраняются в tinyint(1).

Полученная даталогическая схема БД в нотации Мартина изображена на рисунке 11.

Ниже представлено подробное описание приведенных таблиц и их полей.

Таблица Users (пользователи):

- id — первичный ключ;
- nickname — псевдоним;
- is_admin — признак администратора.

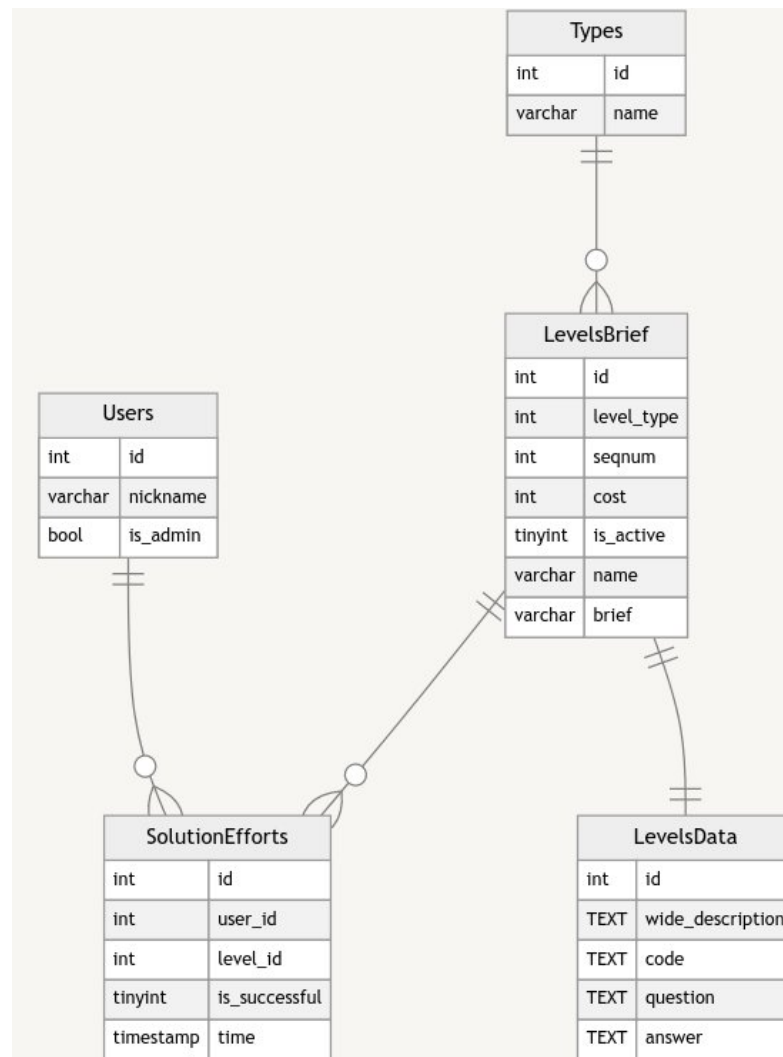


Рисунок 11 — Дatalogическая схема БД

Таблица LevelsBrief (краткая информация о заданиях):

- id — первичный ключ;
- level_type — тип задания;
- seqnum — порядковый номер задания в списке (может повторяться у «заархивированных» заданий);
- cost — количество баллов, начисляемых за решение задания;
- is_active — признак активности задания (если is_active = 0, задание считается «заархивированным»);
- name — название задания;
- brief — краткое описание задания.

Таблица LevelsData (подробная информация о заданиях):

- id — первичный ключ, совпадает с id задания в LevelsBrief;

- wide_description — развернутое описание задания;
- code — листинг исходного кода на Verilog, который может быть приложен к заданию;
- question — закодированные условия задания;
- answer — закодированный ответ на задание.

Таблица Types (типы заданий):

- id — первичный ключ;
- name — название типа задания.

Таблица SolutionEfforts (попытки решения заданий):

- id — первичный ключ;
- user_id — id пользователя;
- level_id — id задания;
- is_successful — признак успешного прохождения задания;
- time — дата и время прохождения задания.

2.2.2 Описание структур данных заданий и ответов

Так, как в разрабатываемой подсистеме используются задания различных типов, которые необходимо проверять автоматически, было решено хранить информацию об условиях и ответах на каждое задание в закодированном виде в одном поле БД (это позволило использовать реляционную модель, позволяющую, например, удобным образом анализировать статистику выполнения заданий).

Разработанная подсистема поддерживает работу с тремя типами заданий:

- тесты с выбором одного варианта ответа;
- тесты с множественным выбором;
- задания на описание устройства на Verilog.

Все задания и ответы сохраняются в нотации JSON.

Условия задания с выбором одного ответа содержат заголовок задания (caption) и массив ответов (answers), в котором каждый ответ имеет поля с

текстом варианта ответа (text) и подсказкой, которая будет показана пользователю, если ответ неверен (hint).

Формат описания задания с выбором одного ответа приведен на рисунке 12.

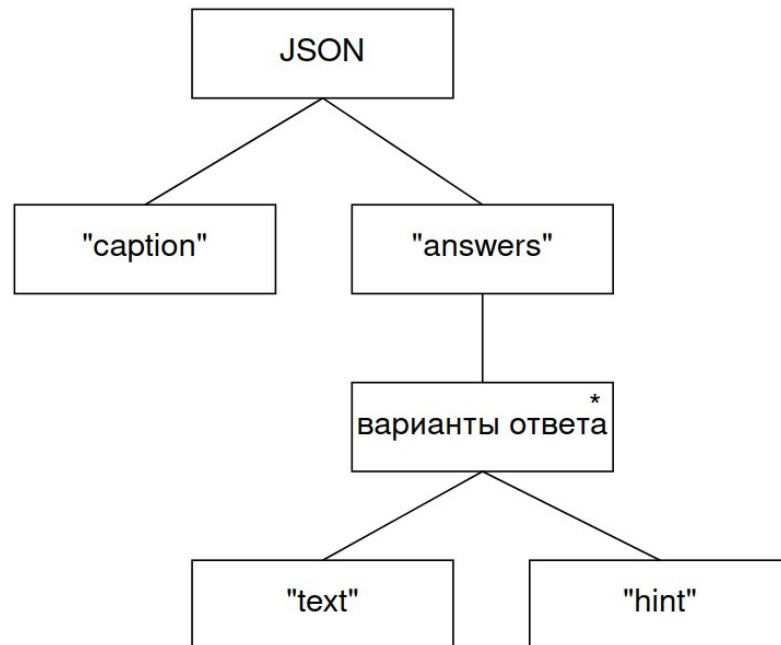


Рисунок 12 — Формат описания задания с выбором одного ответа
Пример описания задания с одним ответом приведен в листинге 1.

Листинг 1 — Пример описания задания с одним ответом

```
// условия задания с выбором одного варианта ответа

{
  "caption": "Основная функция сумматора",
  "answers": [
    {"text": "Умножение", "hint": "Название говорит само за себя"},
    {"text": "Вычитание", "hint": "Перечитай главу"},
    {"text": "Сложение", "hint": "Все верно"}
  ]
}

// ответ на задание с выбором одного варианта ответа

{"correct_answer_id":2}
```

Условия задания с выбором нескольких вариантов ответа хранятся в аналогичном формате, но в них отсутствует поле hint.

Примеры записи условия для заданий с множественным выбором приведен в листинге 2.

Листинг 2 — Пример описания задания с множественным выбором

```
// условия задания с выбором нескольких вариантов ответа

{
  "caption": "Типы переменных в verilog",
  "answers": [
    "reg",
    "wire",
    "mem"
  ]
}

// ответ на задание с выбором нескольких вариантов ответа

{
  "correct_answers": [
    true,
    true,
    false
  ]
}
```

В случае задания на описание устройства с помощью языка Verilog, в поле LevelsData.question заносится код теста устройства на языке Verilog (т.н. «testbench», см. приложение Д), а в поле LevelsData.answer — описание временной диаграммы корректно описанного устройства в формате wavedrom (см. раздел «Генератор wavedrom-диаграмм»).

2.3 Проектирование микросервисов

2.3.1 Микросервис взаимодействия с БД

Для реализации CRUD-операций с данным, хранящимися в БД, был реализован микросервис взаимодействия с БД.

Логика работы с каждой из таблиц базы данных инкапсулирована в отдельный класс, каждый из таких классов работает с БД через класс соединения с БД, который в свою очередь использует драйвер СУБД MySQL (рисунок 13).

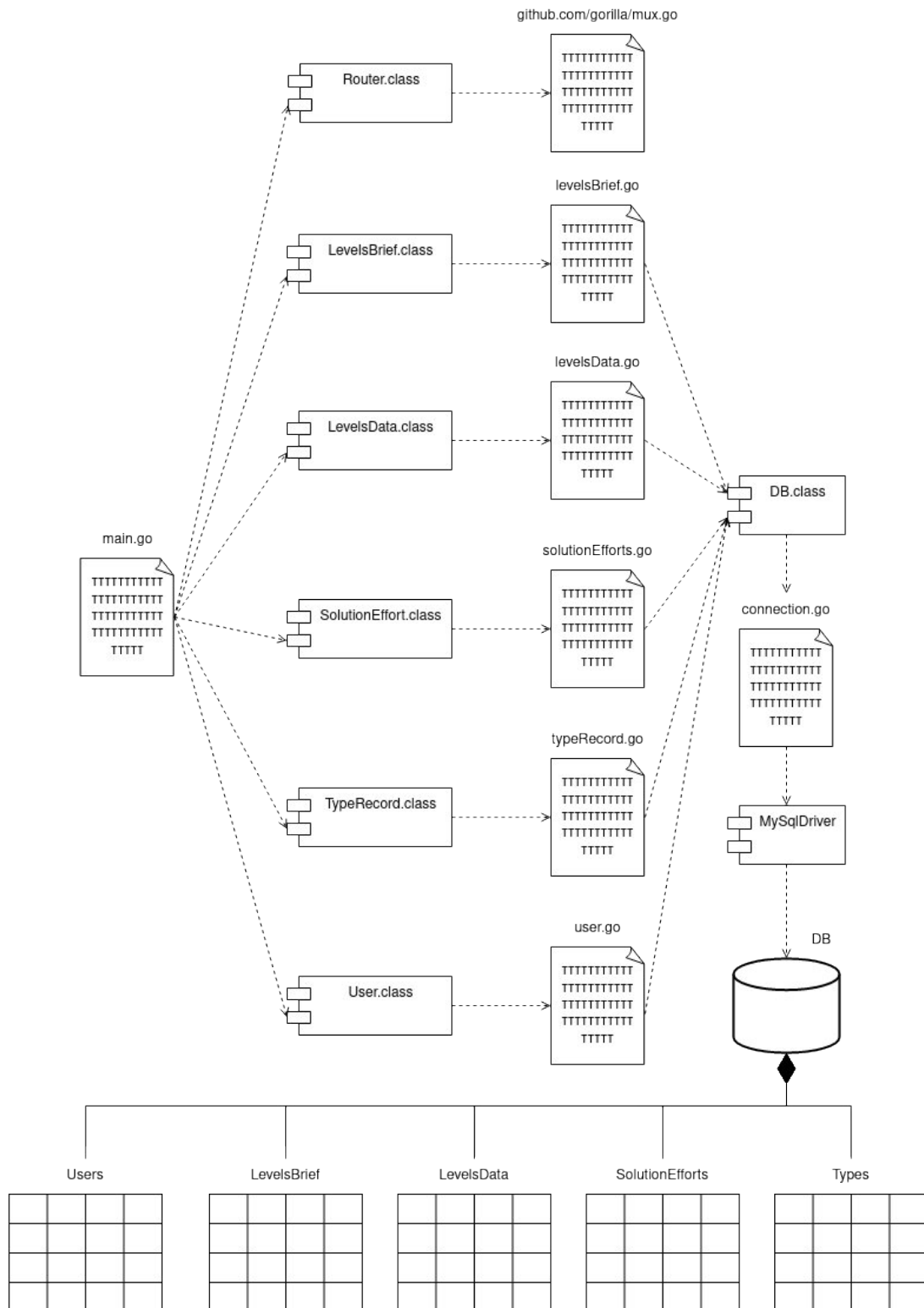


Рисунок 13 — Диаграмма компоновки микросервиса взаимодействия с БД

На рисунке 14 представлена диаграмма классов описываемого микросервиса.

Классы LevelsBrief, LevelsData, SolutionEffort, TypeRecord и User реализуют взаимодействие с БД и воплощают в себе сущности предметной области.

Класс MetaInfo содержит поля ObjType (сущность, над которой выполняется операция) и Action (тип операции).

Классы с префиксом «Rf» (сокращение от «Request Frame») позволяют разобрать входные сообщения, разделив метаинформацию и данные о сущности предметной области.

Интерфейсы IReadable, IUpdatable и т.п. позволяют взаимодействовать с любым типом сущностей по одному и тому же алгоритму [8].

Использование типа interface для ResponseFrame.Data (данные ответного сообщения) так же позволяет записывать в это поле данные об объекте любого класса.

Примечание: любой класс в Golang является реализацией interface, однако не все отношения реализации показаны на диаграмме классов с целью ее упрощения.

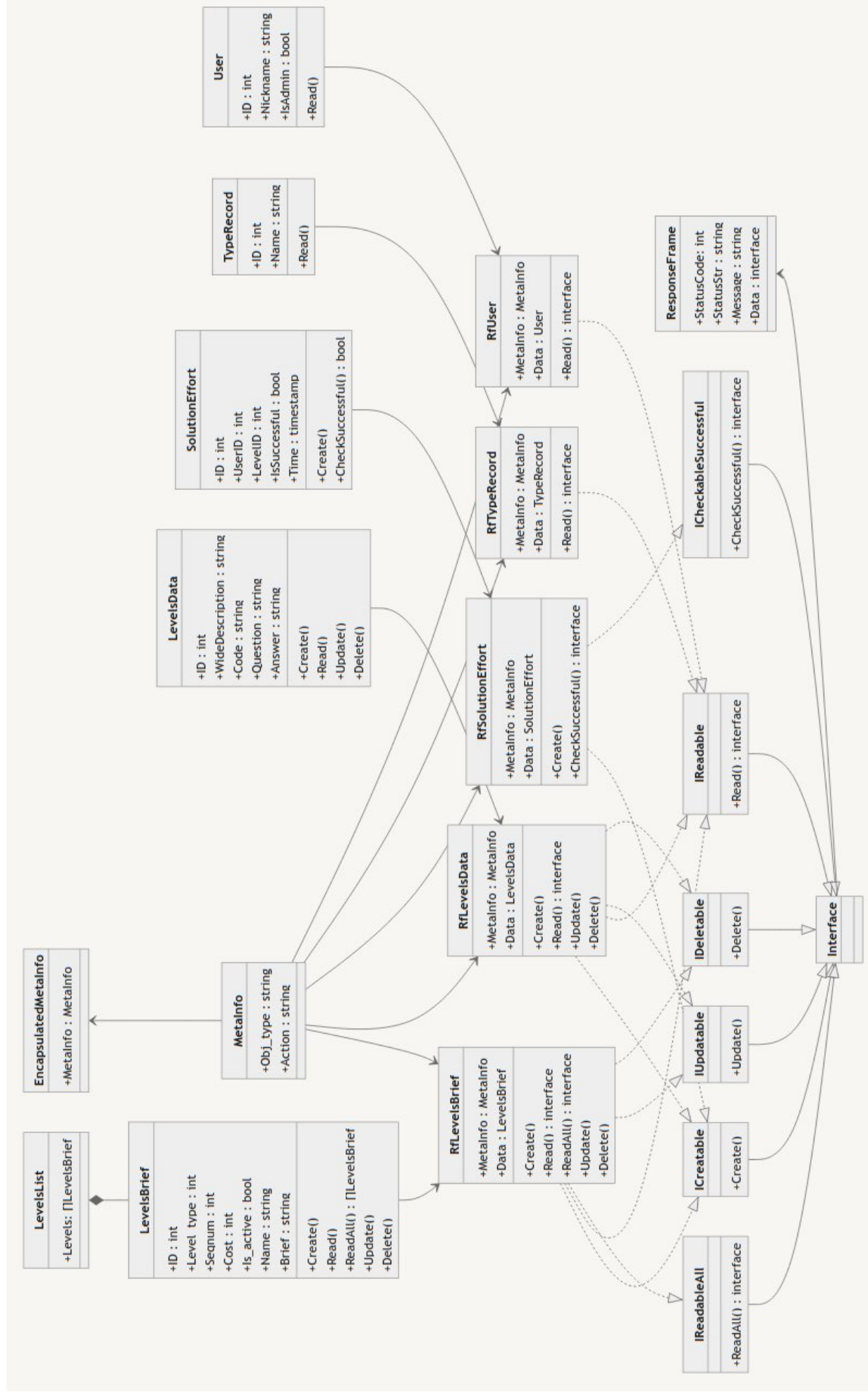


Рисунок 14 — Диаграмма классов для работы с БД

Фрагмент программного кода, иллюстрирующий работу с интерфейсами приведен в листинге 3.

Листинг 3 — Фрагмент программного кода микросервиса взаимодействия с БД.

```
var reqFrame EncapsulatedMetaInfo
// преобразование тела HTTP-запроса в объект класса EncapsulatedMetaInfo:
err = json.Unmarshal(reqBody, &reqFrame)
if err != nil {
    /* обработка ошибок */
}

var data interface{} // создание объекта базового класса
// выбор класса, к которому произойдет обращение
if reqFrame.MetaInfo.ObjType == "levels_brief" {
    data = &RfLevelsBrief{} // присвоение ссылки на пустой экземпляр класса
} else if reqFrame.MetaInfo.ObjType == "levels_data" {
    data = &RfLevelsData{}
} else if
    /* ... */
} else {
    panic("Unknown Obj Type")
}

// преобразование тела HTTP-запроса в объект выбранного класса
err = json.Unmarshal(reqBody, data)
if err != nil {
    /* обработка ошибок */
}
if reqFrame.MetaInfo.Action == "create" {
    data.(ICreatable).Create() // обращение к методу класса через интерфейс
} else if reqFrame.MetaInfo.Action == "read" {
    // обращение к методу класса через интерфейс и запись данных в поле типа interface
    response.Data = data.(IReadable).Read()
} else if
    /* ... */
} else {
    panic("Unknown Action")
}
```

2.3.2 Микросервис синтеза устройств (синтезатор)

Для симуляции (получения временных диаграмм работы) и синтеза (получения списка электрических соединений) устройств, описанных на языке Verilog используют специальное программное обеспечение, которое может называться «симулятором», «синтезатором» или «компилятором» (последний термин менее точен, но более интуитивно понятен).

Одним из таких синтезаторов является Icarus Verilog. Достоинствами данного программного решения являются:

- малый размер исполняемого файла;
- наличие консольного режима работы (удобно вызывать из программного кода через библиотеки для работы с операционной системой);
- распространение по свободной лицензии (GNU GPL).

В силу перечисленных выше свойств, Icarus Verilog был выбран в качестве синтезатора, используемого в данной работе.

Выбранный синтезатор был использован в составе микросервиса синтеза устройств, обеспечивающего управление файлами, взаимодействие с сетью и синтез устройств с помощью Icarus Verilog.

Обработка каждого задания осуществляется в несколько этапов, за каждый из которых отвечает свой программный компонент:

- получение http-запроса на симуляцию устройства (класс Router);
- сохранение полученных исходных кодов устройства и теста в файловой системе (OsLib, наличие user_id и level_id позволяет значительно снизить риск коллизии файлов);
- получение временной диаграммы работы устройства (в формате *.vcd) с помощью IcarusVerilog;
- отправка http-ответа, содержащего код временной диаграммы.

Диаграмма компоновки микросервиса показана на рисунке 15.

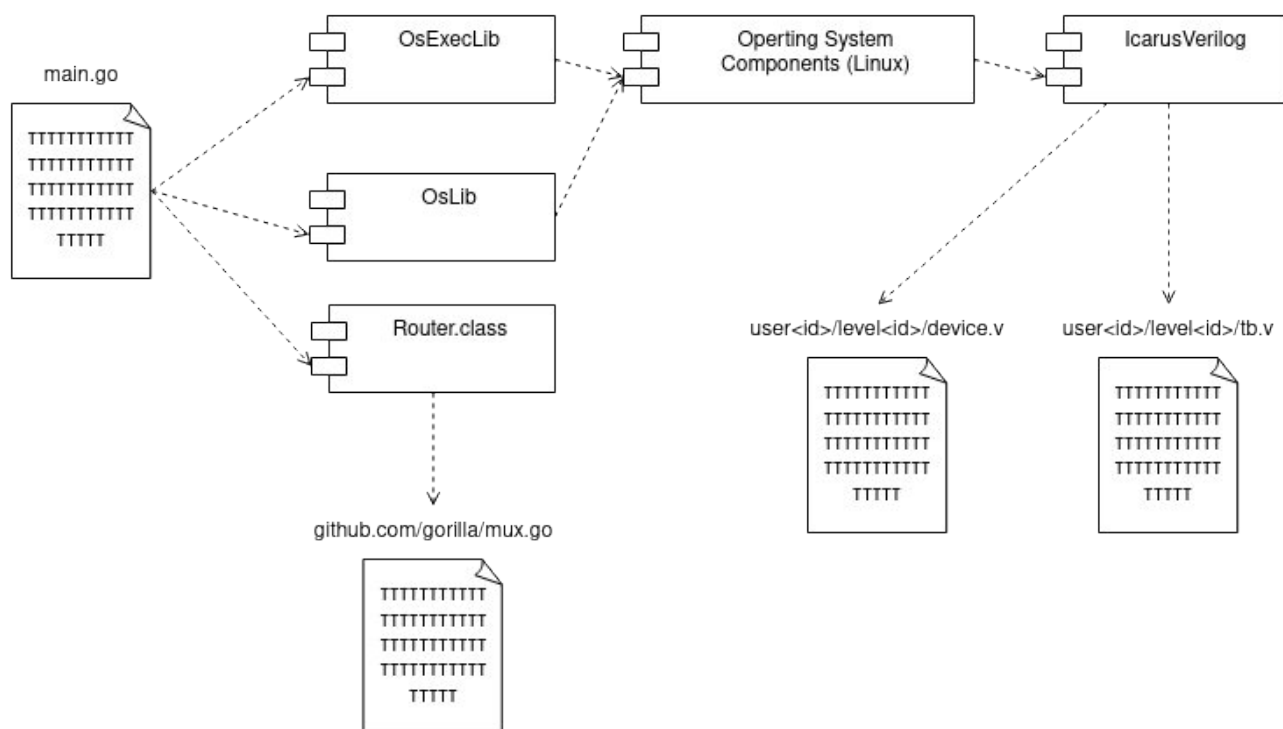


Рисунок 15 — Диаграмма компоновки микросервиса синтеза устройств

2.3.3 Микросервисы для работы с временными диаграммами

Изначально микросервис синтеза устройств в ходе тестирования работы устройства формирует временную диаграмму в формате *.vcd (Приложение Е). Данный формат крайне неудобен, как для анализа в сравнении с эталонной временной диаграммой, так и для генерации графического представления временной диаграммы в рамках веб-приложения.

Для преобразования временных диаграмм к более удобному для дальнейшей обработки формату был реализован микросервис разбора временных диаграмм.

Его исходный код написан на Python с применением библиотеки PyDigitalWaveTools. Данная библиотека преобразует временную диаграмму в формате *.vcd в формат JSON-PyDigitalWaveTools согласно алгоритму, заложенному автором библиотеки. Диаграмма Джексона, описывающая этот формат представлена на рисунке 16 [9].

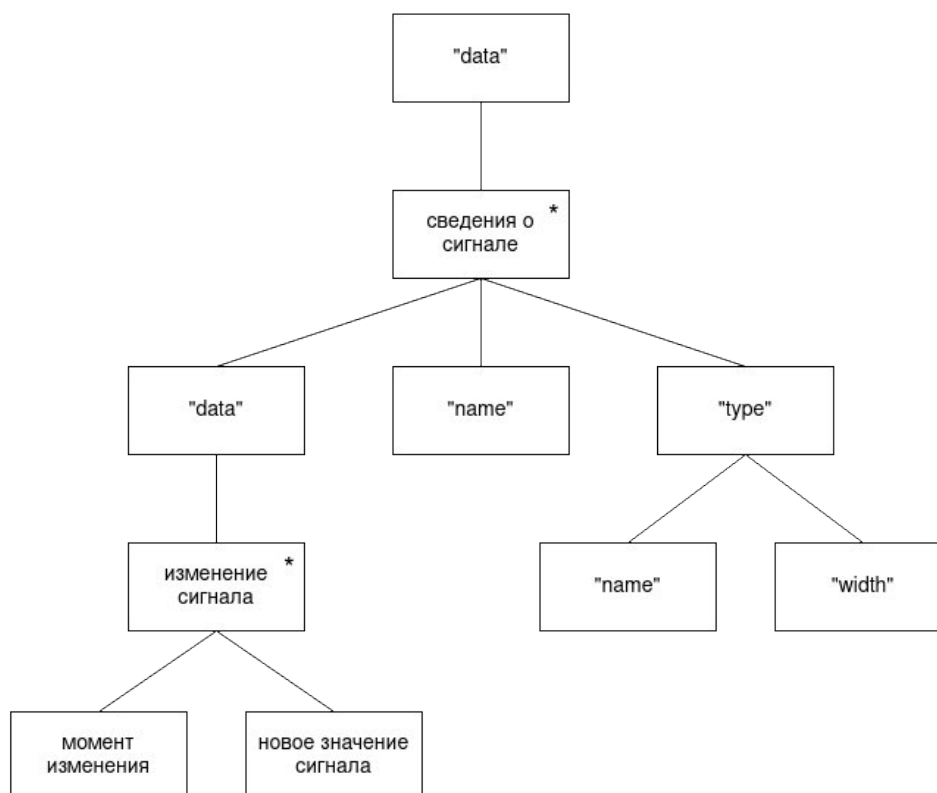


Рисунок 16 — Формат временных диаграмм в PyDigitalWaveTools

В нем поле `data.name` — имя сигнала, `data.type.name` — название типа сигнала (комбинационный или регистровый), `data.type.width` — разрядность сигнала. «Момент изменения» — количество элементарных отрезков времени (их размер определяется в момент написания теста для устройства) от начала отсчет до изменения сигнала.

Пример описания сигнала в этом формате приведен в листинге 4.

Листинг 4 — пример описания временной диаграммы в PyDigitalWaveTools

```

{
  "data": [
    {
      "data": [
        [0, "b0"],
        [100, "b1"]
      ],
      "name": "Sum",
      "type": {
        "name": "wire",

```

```

        "width": 4
    }
},
/* ... */
]
}

```

Формат PyDigitalWaveTools намного более удобен для сравнения с эталонной временной диаграммой (в том же формате) и анализа несоответствий, однако алгоритм визуализации для этого формата пришлось бы реализовать самостоятельно.

Вместо этого было решено реализовать микросервис генерации временных диаграмм wavedrom, который преобразовал бы временные диаграммы из формата PyDigitalWaveTools в формат движка Wavedrom [10]. Данный движок позволяет визуализировать временные диаграммы посредством http-запроса, содержащего описание сигнала, к специальному интернет-сервису.

Описание формата для движка Wavedrom в нотации Джексона приведено на рисунке 17.

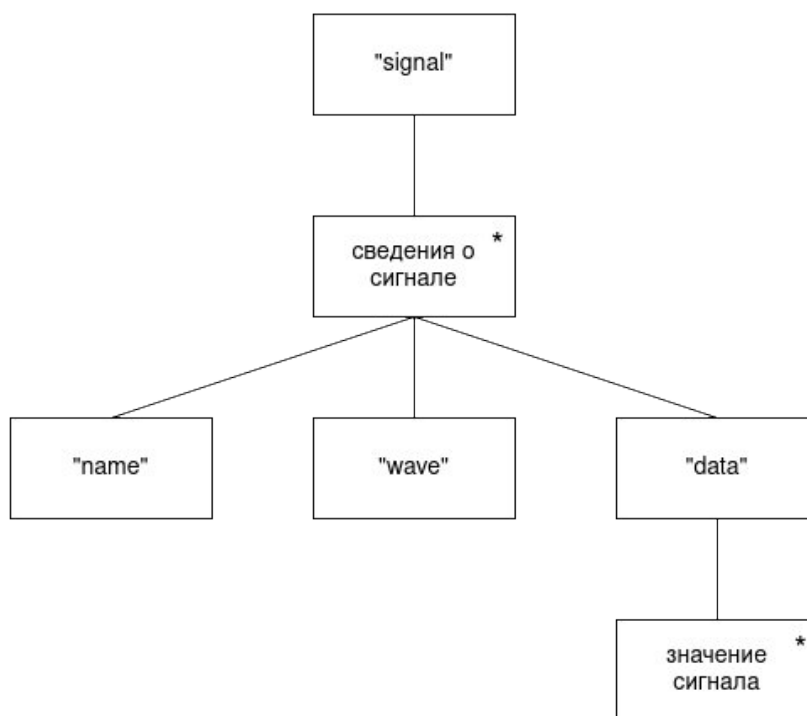


Рисунок 17 — Формат временных диаграмм для движка Wavedrom

Поля структуры имеют значение, описанное ниже:

- signal — массив всех сигналов временной диаграммы;
- name — имя сигнала;
- wave — форма сигнала (для каждого такта может иметь значения: «0», «1», «x», «z», «.» — сохранить предыдущее, «|» — разрыв, «=» — обратиться к очередному элементу «data»);
- data — массив, содержащий строковые значения сигнала (можно, например, отобразить большое число для многоразрядной шины).

Пример описания временной диаграммы в формате движка Wavedrom приведен в листинге 5.

Листинг 5 — Описание временной диаграммы в формате движка Wavedrom

```
{signal: [  
  {name: 'clk', wave: 'p.....|...'},  
  {name: 'dat', wave: 'x.345x|=x', data: ['0x16', '0xAA', '0x07', '0x11']},  
  {name: 'req', wave: '0.1..0|1.0'},  
  {},  
  {name: 'ack', wave: 'z.....|01.'}  
]}
```

Визуализация данной временной диаграммы приведена на рисунке 18.

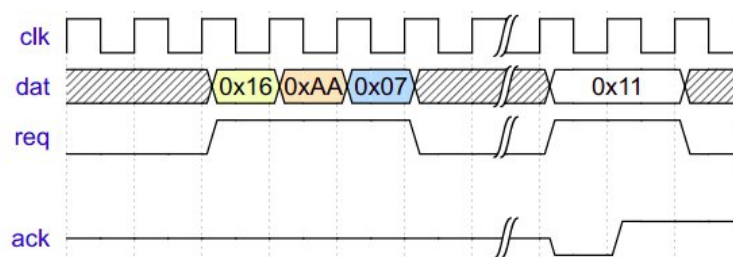


Рисунок 18 — Визуализация временной диаграммы

Основной функцией «Генератора wavedrom-диаграмм» является функция `parseValues`, программный код которой приведен в листинге 6.

Листинг 6 — Программный код функции `parseValues`

```
func (vcd_frame VCD_Struct) parseValues(end_scale int, width_scale int) (map[string]string,  
map[string][]string) {  
  // значения сигналов, e.g.: {"a": ["0x10", "0x35", "0xA1"], "b": ["0x03", "0x0F"]}  
  var parsedData = map[string][]string{
```



```

// форма сигналов, e.g.: {"a": "1...0.....1..", "b": "0....=....=.."}
var parsedWaves = map[string]string{}
// отсортированные моменты изменения всех сигналов
timings := vcd_frame.getSortedTimings()
tick_amount := findGCD(timings) // НОД момента изменения сигнала
end_time := timings[len(timings)-1] + tick_amount*end_scale

for i := 0; i < end_time/tick_amount; i++ { // проход по всем моментам дискретизации
    for _, single_signal := range vcd_frame.Signal { // проход по всем сигналам диаграммы
        fl_change := false // признак изменения сигнала в этом моменте времени
        name := single_signal.Name
        fl_single_wire := true // признак однобитного сигнала
        if single_signal.Type.Width > 1 {
            name += "[0:" + strconv.Itoa(single_signal.Type.Width-1) + "]"
            fl_single_wire = false
        }
        // проход по всем точкам дискретизации
        for _, data_value := range single_signal.Data {
            // изменился ли сигнал в рассматриваемой точки дискретизации?
            if int(math.Round(data_value[0].(float64))) == i*tick_amount {
                if fl_single_wire {
                    parsedWaves[name] += data_value[1].(string)
                } else {
                    parsedData[name] = append(parsedData[name], data_value[1].(string))
                    parsedWaves[name] += "="
                }
                fl_change = true
                break
            }
        }
        if !fl_change {
            parsedWaves[name] += "."
        }
        // масштабирование ширины сигнала на диаграмме
        for j := 1; j < (vcd_frame.getMaxValueWidth()*width_scale)/2; j++ {
            parsedWaves[name] += "."
        }
    }
}
return parsedWaves, parsedData
}

```

2.3.4 Микросервис анализа статистики

Для работы со статистикой прохождения заданий было решено реализовать отдельный микросервис, структура которого была бы аналогична структуре микросервиса взаимодействия с БД (диаграмма компоновки представлена на рисунке 19).

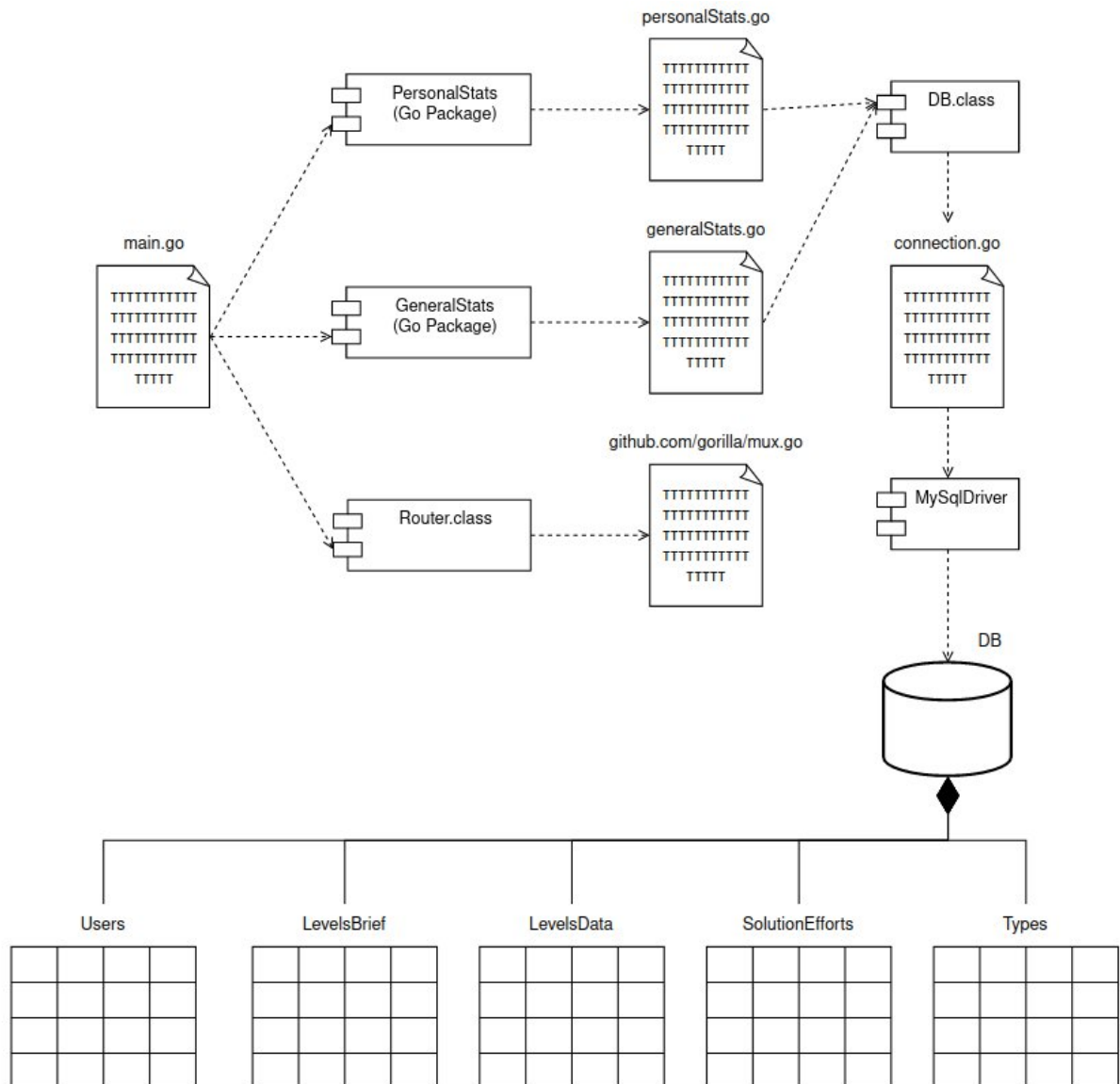


Рисунок 19 — Диаграмма компоновки микросервиса анализа статистики

Микросервис реализует, два вида запросов — запросы, получающие персональную статистику пользователя (требуется `user_id`), и запросы, получающие обобщенную статистику для всех пользователей. Полученные данные возвращаются в формате JSON.

Запросы, получающие персональную статистику:

- прогресс по курсу — количество полученных баллов и решенных заданий, статус прохождения курса (<80% баллов от максимума — «not_passed», 80-90% — «passed», >90% — «awesome»);
- статус прохождения каждого задания (информация об уровне из таблицы LevelsBrief и признак «is_solved» для каждого задания);
- среднее число неправильных попыток на задание;
- общее число попыток и правильных решений за последний месяц;
- Дата решения первого и последнего решенного задания.

Запросы, получающие обобщенную статистику:

- количество верных решений для каждого задания;
- среднее число ошибок в каждом задании;
- распределение количества пройденных заданий в зависимости от числа их решений (абсцисса — количество решений n , ордината — число заданий которые решили n раз);
- среднее число предоставленных пользователями решений по месяцам;
- топ 10 активных пользователей за последний месяц.

2.3.5 Микросервис анализа решений (анализатор)

Анализатор представляет собой микросервис, задачей которого является проверка пользовательских ответов, а также предоставление подсказок в случае допущения пользователем ошибки.

Анализатор способен работать с тремя типами заданий:

- тест с выбором одного варианта ответа — анализатор сопровождает неправильный ответ текстовой подсказкой;
- тест с выбором нескольких вариантов ответа — анализатор сопровождает неправильный ответ информацией о наличии/отсутствии ложноположительных/ложноотрицательных вариантов;

- задание на описание устройства на языке Verilog — анализатор приводит список отсутствующих выходных сигналов и список сигналов, поведение которых не соответствует ожидаемому.

Для универсализации алгоритма обработки решений используется полиморфизм (рисунок 20).

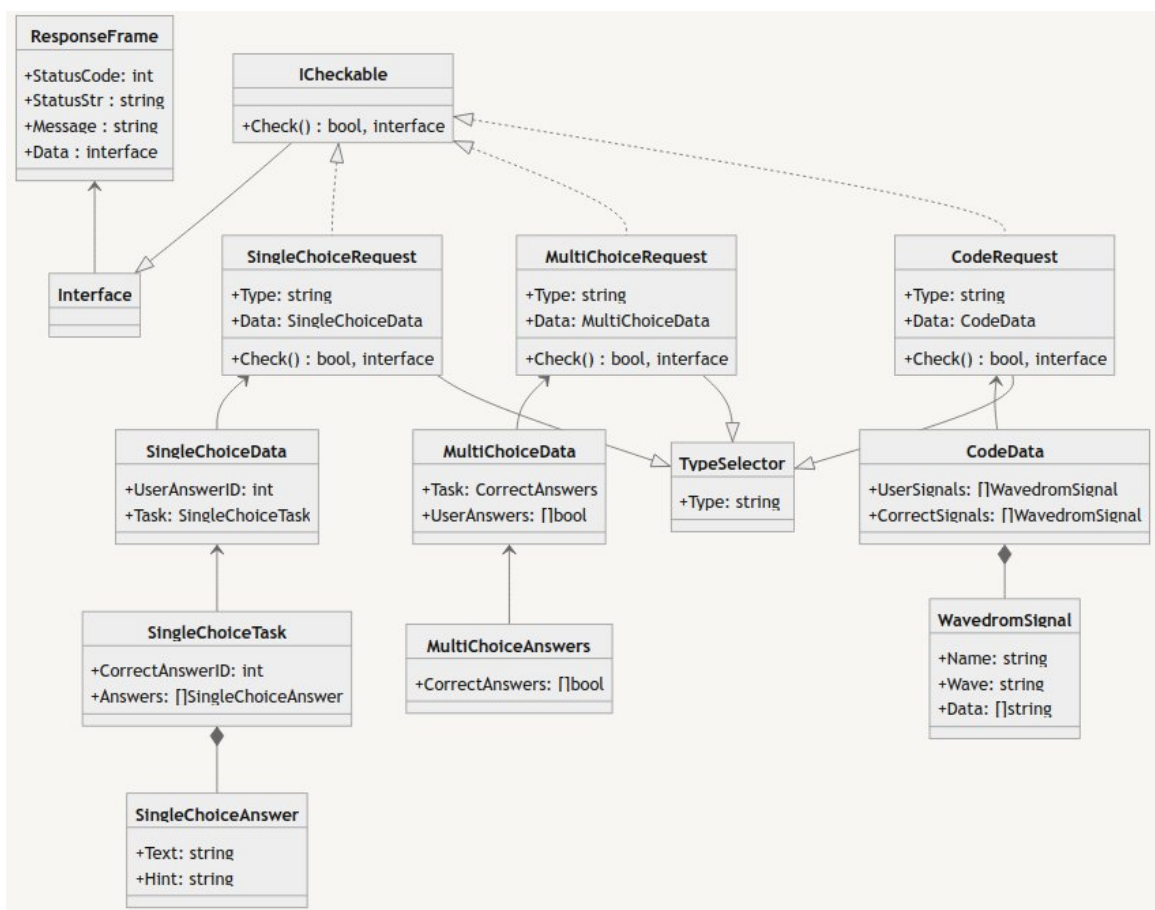


Рисунок 20 — Диаграмма классов анализатора решений

Исходный код этого микросервиса приведен в приложении Г.

2.4 Выводы

В рамках конструкторской части была спроектирована структура компонентов и спроектированы сами компоненты подсистемы тестирования знаний языков описания аппаратуры: подсистема взаимодействия с БД, синтезатор устройств, преобразователи формата временных диаграмм, анализатор решений и подсистема анализа статистики.

Спроектированная подсистема реализует функции и варианты использования, упомянутые в исследовательской части.

3 Разработка технологии тестирования

3.1 Выбор подходов и методов тестирования

Процесс разработки программного обеспечения в том виде, как оно определяется в современной модели жизненного цикла программного обеспечения предполагает три стадии тестирования [11]:

- автономное тестирование компонентов ПО;
- комплексное тестирование разрабатываемого ПО;
- системное или оценочное тестирование на соответствие основным критериям качества.

Для проведения автономного и комплексного тестирования необходимо сформировать тестовые наборы, опираясь на структурный или функциональный подход.

Структурный подход базируется на том, что известна структура тестируемого ПО, в том числе его алгоритмы. Тесты строят так, чтобы обеспечить максимальное покрытие исходного кода.

Функциональный подход основывается на том, что структура ПО не известна. В этом случае тесты строят, опираясь на функциональные спецификации. Тесты строят на базе различных способов декомпозиции множества данных.

Разработанное ПО включает в себя разнородные алгоритмы, для всестороннего тестирования которых с помощью структурного подхода понадобились бы значительные затраты времени на изучение исходного кода, разработку большого числа тестов и загрузок. По этой причине было решено использовать функциональный подход, который позволил бы значительно сократить время на разработку тестов, обеспечивая при этом тестирование всей необходимой функциональности [12].

В качестве оценочного тестирования согласно ТЗ было выбрано нагрузочное тестирование.

3.2 Разработка плана автономного тестирования

В качестве основного метода, используемого для разработки функциональных тестов, был использован метод эквивалентного разбиения.

Все задания, проверяемые анализатором решений, принадлежат к одному из следующих классов эквивалентности:

- тест с выбором одного ответа;
- тест с выбором множества ответа;
- задания на написания программы.

Для каждого из выделенных классов заданий можно выделить 3 класса решений:

- правильное решение задания;
- неправильное решение задания;
- решение, закодированное с нарушением формата.

На основе такого эквивалентного разбиения были составлены тесты, представленные в таблице 4.

Таблица 4 — Планируемые тесты анализатора

Номер теста	Название теста	Описание теста	Ожидаемый результат
1	single correct positive	Отправить на проверку одновариантный тест с ID ответа равным ID верного ответа	Признак is_correct = True
2	single correct negative	Отправить на проверку одновариантный тест с ID ответа не равным ID верного ответа	Признак is_correct = False

Продолжения таблицы 4

3	single error overflow	Отправить на проверку одновариантный тест с ID ответа больше максимального допустимого	Признак is_correct = False
4	multi correct positive	Отправить на проверку многовариантный тест с верными ответами	Признаки is_correct = True, false_positive = False, false_negative = False
5	multi correct false positive	Отправить на проверку многовариантный тест с ложноположительными ответами	Признаки is_correct = False, false_positive = True, false_negative = False
6	multi correct false negative	Отправить на проверку многовариантный тест с ложноотрицательными ответами	Признаки is_correct = False, false_positive = False, false_negative = True
7	multi error size mismatch	Отправить на проверку многовариантный тест с разной длиной массива ответов пользователя и эталонного ответа	Ошибка "answer size mismatch"
8	code correct positive	Отправить на проверку корректно выполненное задание на написание кода	Признак is_correct = True
9	code correct negative	Отправить на проверку некорректно выполненное задание на написание кода	Признак is_correct = False, возвращен список несовпадающих с эталоном сигналов

Микросервис взаимодействия БД работает с данными (метаданными) 4 классов (для каждой из сущностей):

- данные для создания экземпляра сущности;
- метаданные для чтения экземпляра сущности;
- данные для изменения экземпляра сущности;
- метаданные для удаления экземпляра сущности.

В каждом из этих классов могут содержаться следующие подклассы:

- корректные данные (метаданные);
- данные в некорректном формате (отсутствуют поля и т.п.);
- метаданные с несуществующим ID.

Так как классы эквивалентности одинаковы для всех сущностей, в таблице 5 приведены примеры тестов для сущности LevelsBrief. Тесты для других сущностей аналогичны.

Таблица 5 — Планируемые тесты микросервиса взаимодействия с БД для сущности LevelsBrief

Номер теста	Название теста	Описание теста	Ожидаемый результат
1	correct read level brief	Отправить запрос на считывание экземпляра сущности LevelsBrief с заданным ID из БД	Строка из БД с информацией о сущности LevelsBrief с заданным ID
2	correct create level brief	Отправить запрос на создание экземпляра сущности LevelsBrief, считать информацию о ней	Строка в БД с заданными значениями полей

Продолжение таблицы 5

3	correct update level brief	Отправить запрос на изменение экземпляра сущности LevelsBrief с заданным ID, считать информацию о ней	Строка в БД с новыми значениями полей
4	correct delete level brief	Отправить запрос на удаление экземпляра сущности LevelsBrief с заданным ID, считать информацию о ней	Поле is_archived = True
5	error create level invalid format	Отправить запрос на создание экземпляра сущности LevelsBrief, считать информацию о ней	Ошибка "invalid data format"
6	error read level invalid id	Отправить запрос на считывание экземпляра сущности LevelsBrief с несуществующим ID из БД	Ошибка "LevelsBrief entity with ID=<id> does not exist"

Для преобразователей временных диаграмм и синтезатора данные можно разделить на корректные и некорректные. Эти тесты приведены в таблицах 6-7.

Таблица 6 — Планируемые тесты микросервиса разбора временных диаграмм

Номер теста	Название теста	Описание теста	Ожидаемый результат
1	correct positive	Отправить запрос на преобразование временной диаграммы в формате VCD с корректными данными	JSON с описанием временной диаграммы в заданном формате

Продолжение таблицы 6

2	error in vcd	Отправить запрос на преобразование временной диаграммы в формате VCD с произвольными символами вместо коррентных данных	Ошибка "vcd parsing error"
---	--------------	---	----------------------------

Таблица 7 — Планируемые тесты микросервис генерации временных диаграмм wavedrom

Номер теста	Название теста	Описание теста	Ожидаемый результат
1	correct positive	Отправить запрос на преобразование временной диаграммы в формате PyDigitalWaveTools	Временная диаграмма в формате wavedrom
2	error format	Отправить запрос на преобразование временной диаграммы с произвольными данными	Ошибка "invalid data format"

Для синтезатора в классе некорректных данных были выделены подклассы:

- данные с ошибкой в исходном коде устройства;
- данные с ошибкой в исходном коде тестов;
- данные без необходимой директивы "\$dumpvars".

Соответствующие тесты приведены в таблице 8

Таблица 8 — Планируемые тесты синтезатора

Номер теста	Название теста	Описание теста	Ожидаемый результат
1	correct positive	Отправить корректный исходный код описания устройства и тестов	Временная диаграмма в формате VCD
2	error in device	Отправить некорректный исходный код описания устройства и корректный код тестов	Ошибка "synthethis error"
3	error in testbench	Отправить корректный исходный код описания устройства и некорректный код тестов	Ошибка "simulation error"
4	error no dumpvars	Отправить корректный исходный код описания устройства и тестов, но без директивы dumpvars	Ошибка "testbench without \$dumpvars"

Составленные автономные тесты позволяют обеспечить высокую степень покрытия функций компонентов разработанного ПО.

3.3 Разработка плана комплексного тестирования

Так как большинство базовых функций разработанной подсистемы тестирования знания было протестировано в режиме автономного тестирования, для проведения комплексного тестирования будет достаточно проверить пользовательские сценарии проверки задания, создания/изменения задания, возможность обращения к микросервисам анализа статистики и работы с БД.

Тесты для комплексного тестирования приведены в таблице 9.

Таблица 9 — Планируемые тесты основного микросервиса

Номер теста	Название теста	Описание теста	Ожидаемый результат
1	correct proxy crud	Запросить данные о пользователе с ID = 1 из БД	Данные о пользователе с ID = 1 из БД
2	correct proxy stats	Запросить статистику прохождения заданий на основе тестовых данных	Показатели статистики соответствуют предварительно рассчитанным
3	error no user in check	Запросить проверку задания для несуществующего пользователя	Ошибка "user does not exist"
4	error not admin in crud	Запросить изменение задания от имени пользователя, не являющегося администратором	Ошибка "user have no rights to modify levels"
5	correct check	Запросить проверку правильно выполненного задания	Данные о выполнении задания занесены в БД
6	error no level in check	Запросить проверку несуществующего задания	Ошибка "crud-microservice.levelsbrief error"
7	error create levelsdata	Запросить создание задания на программирование, указав некорректный исходный код описания устройства	Ошибка "device synthesis error"
8	correct create levelsdata	Запросить создание задания на программирование	Задание добавлено в БД

3.4 Выбор языка программирования и библиотек для функционального тестирования

Так как написание тестов на Golang требует значительного времени и такие тесты сложнее поддерживать в силу непопулярности языка среди тестировщиков, было решено тестировать разработанные микросервисы, предварительно запустив их (см. приложение Б) и обращаясь к ним по протоколу HTTP. Такой подход позволил реализовать тесты не привязываясь к языку реализации исходного ПО.

Поскольку Python обладает простым синтаксисом, большим количеством библиотек и популярен среди тестировщиков (рисунок 21), именно он был выбран для реализации тестов [13].

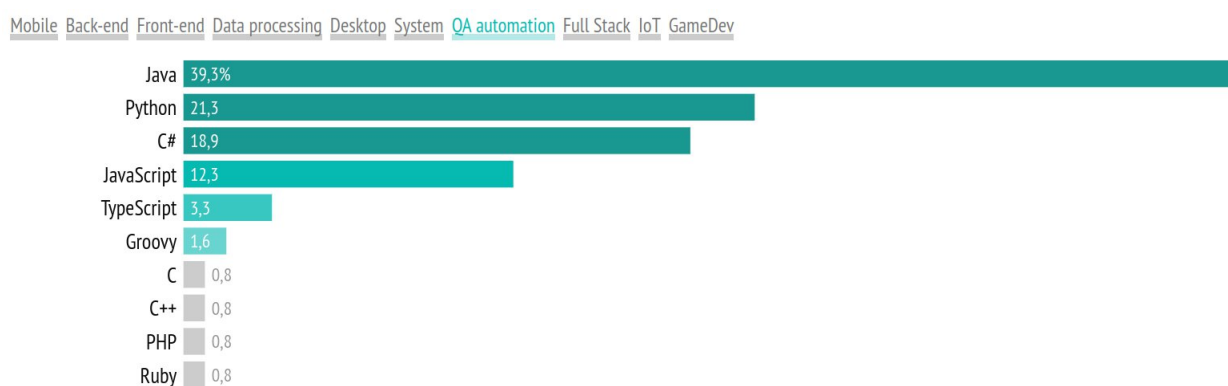


Рисунок 21 — Наиболее популярные языки в области автоматизированного тестирования

В качестве основной библиотеки для тестирования была выбрана библиотека `pytest`, являющаяся одной из наиболее популярных библиотек для автоматизированного тестирования [14].

`Pytest` обладает следующими основными преимуществами [15]:

- меньше повторяющегося кода за счет независимости от API;
- выполнение определенного набора тестов с помощью фильтрации;
- параметризация тестов — запуск одного и того же теста с разными наборами параметров;
- гибкость — архитектура библиотеки основана на плагинах, которые можно установить отдельно;

- полная обратная совместимость с unittest — возможность запуска тестов, написанных на нем;
- выполнение нескольких тестов параллельно;
- установочный код можно использовать повторно.

В дополнение к pytest была использована библиотека allure, формирующая интерактивные отчеты о прохождении тестов. Тесты в allure можно иерархически группировать и сопровождать логами и вложениями. Allure поддерживается не только для Python, но и для Java, JavaScript, Ruby, PHP, .Net и Scala.

Такой широкий набор поддерживаемых языков программирования делает allure (рисунок 22) знакомым многим разработчикам, тестировщикам и менеджерам, что упрощает поддержку тестов [16].

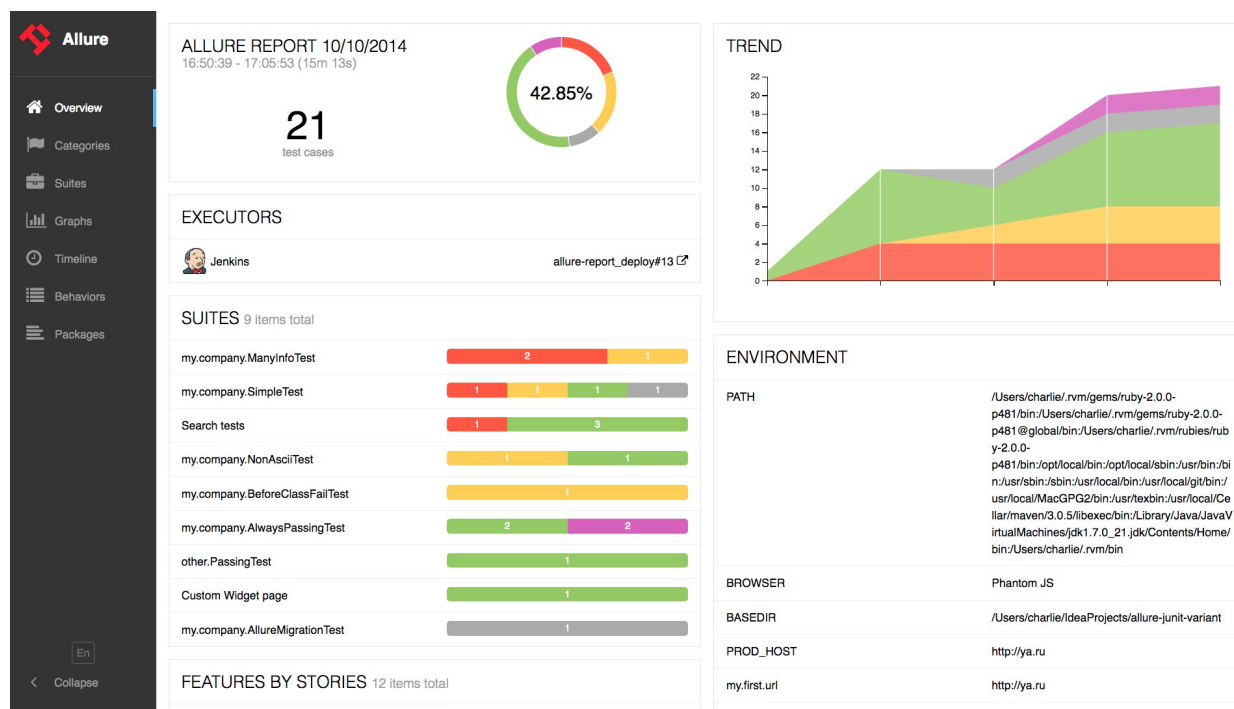


Рисунок 22 — Интерфейс allure

3.5 Реализация и проведение функциональных тестов

Для упрощения написания тестов и генерации отчетов был реализован вспомогательный модуль `utils.py`, отвечающий за отправку http-запросов к микросервисам, проверку http-ответов и их прикрепление к отчетам в allure. Программный код `utils.py` приведен в листинге 7.

Листинг 7 — Программный код utils.py

```
import requests
import allure

@allure.step("Send request") # отправка запроса - новый этап в отчете
def send_request(port, url, payload):
    path = f"http://127.0.0.1:{port}/{url}"
    allure.attach(path, 'Request URL', allure.attachment_type.TEXT)
    # прикрепить данные запроса к отчету
    allure.attach(json.dumps(payload, indent=4, ensure_ascii=False).encode(), 'Request
payload', allure.attachment_type.TEXT)
    resp = requests.post(path, json = payload)
    # прикрепить данные ответа к отчету
    allure.attach(json.dumps(resp.json(), indent=4, ensure_ascii=False).encode(), 'Response
payload', allure.attachment_type.TEXT)
    return resp

@allure.step("Check response [ok]") # проверка ответа без ошибок
def is_ok_response(response):
    allure.attach(json.dumps(response.json(), indent=4, ensure_ascii=False).encode(),
'Response payload', allure.attachment_type.TEXT)
    return response.status_code == 200 \
        and response.json()["status_code"] == 200 \
        and response.json()["status_str"] == "ok"

@allure.step("Check response [error]") # проверка ответа при ошибке
def is_error_response(response):
    allure.attach(json.dumps(response.json(), indent=4, ensure_ascii=False).encode(),
'Response payload', allure.attachment_type.TEXT)
    return response.status_code != 200 \
        and response.json()["status_code"] != 200 \
        and response.json()["status_str"] == "error"

def ordered_json(obj): # упорядочивание JSON для последующего сравнения
    if isinstance(obj, dict):
        return sorted((k, ordered_json(v)) for k, v in obj.items())
    if isinstance(obj, list):
        return sorted(ordered_json(x) for x in obj)
    else:
        return obj
```

Помимо `utils.py`, были реализованы вспомогательные модули `settings.py` и `consts.py`, которые определяют порты по которым доступны микросервисы и данные для HTTP-запросов в тестах.

Пример реализации простейшего теста (проверка работы анализатора для неправильно решенного пользователем теста с одним вариантом ответа) приведен в листинге 8.

Листинг 8 — Пример реализации теста

```
import utils.settings as settings
import utils.utils as utils
from utils.consts import *

import allure
import copy

# < ...>

# определение иерархии и описания в отчете
@allure.description("Test for singlechoice wrong-answered task")
@allure.epic("Unit-testing")
@allure.story("Analyzer")
def test_single_correct_negative():
    # считывание данных при правильном ответе из констант
    payload = copy.deepcopy(Analyzer.single_valid_positive)
    # изменение ID выбранного ответа
    payload["data"]["user_answer_id"] = 1
    # http-запрос
    resp = utils.send_request(settings.ANALYZER_PORT,
                              "check", payload)
    # http-ответ успешен?
    assert utils.is_ok_response(resp)
    # http-ответ задание решено неверно?
    assert resp.json()["is_correct"] == False
    # возвращена ожидаемая подсказка?
    assert resp.json()[
        "data"]["hint"] == payload["data"]["task"]["answers"][1]["hint"]

# < ...>
```


Пример отчета, полученного после выполнения этого теста (и аналогичных ему) приведен на рисунках 23-24.

The screenshot shows the Allure Behaviors report. On the left, a sidebar lists navigation options: Overview, Categories, Suites, Graphs, Timeline, Behaviors (selected), and Packages. The main area displays a table of behaviors with columns for order, name, duration, and status. The status bar shows 0 failed, 0 skipped, 46 passed, and 0 not run. The behaviors are grouped into categories: Integrational testing (Gateway), Unit-testing (Analyzer), CRUD, General stats, Parser, Synthesizer, and Wavedrom. The 'test_single_correct_negative' behavior is highlighted in yellow. To the right, a detailed view of this behavior is shown, including its severity, duration, description, and execution details.

order	name	duration	status
8	test_code_correct_negative	4ms	Passed
9	test_code_correct_positive	3ms	Passed
6	test_multi_correct_false_negative	3ms	Passed
5	test_multi_correct_false_positive	3ms	Passed
4	test_multi_correct_positive	3ms	Passed
7	test_multi_error_size_mismatch	5ms	Passed
1	test_single_correct_negative	5ms	Passed
2	test_single_correct_positive	5ms	Passed
3	test_single_error_overflow	3ms	Passed

test_analyzer#test_single_correct_negative
Passed test_single_correct_negative
 Overview History Retries
 Severity: normal
 Duration: 5ms
 Description: Test for singlechoice wrong-answered task
 Execution
 Set up
 Test body
 Send request 3 parameters, 3 attachments 4ms
 port: 8083
 url: /check
 payload: {type: 'singlechoice_test', data: {user_answer_id: 1, task: {correct_ans...
 Request URL: http://127.0.0.1:8083/check 27 B
 Request payload: 639 B
 Response payload: 170 B
 Check response [ok] 1 parameter, 1 attachment 1ms
 response: <Response [200]>
 Response payload: 170 B

Рисунок 23 — Отчет о результате теста

The screenshot shows the Allure Behaviors report, similar to Figure 23, but with a detailed view of the 'test_single_correct_negative' behavior. The left sidebar and behavior list are the same. The detailed view on the right shows the request and response payloads for the 'check' endpoint.

Request URL
 http://127.0.0.1:8083/check

Request payload

```
{
  "type": "singlechoice_test",
  "data": {
    "user_answer_id": 1,
    "task": {
      "correct_answer_id": 2,
      "answers": [
        {
          "text": "Умножение",
          "hint": "Название говорит само за себя"
        },
        {
          "text": "Вычитание",
          "hint": "Перечитай главу"
        },
        {
          "text": "Сложение",
          "hint": "Все верно"
        }
      ]
    }
  }
}
```

Response payload

```
{
  "status_str": "ok",
  "status_code": 200,
  "message": "checked",
  "is_correct": false,
  "data": {
    "hint": "Перечитай главу"
  }
}
```

Check response [ok] 1 parameter, 1 attachment

Рисунок 24 — Приложения к отчету

После реализации аналогичным образом всех тестов из таблиц 4-9 они были запущены.

Благодаря параллельному запуску тестов через плагин xdist (в данном случае — в 2 потока) удалось выполнить все тесты менее, чем за 600 мс (рисунок 25).

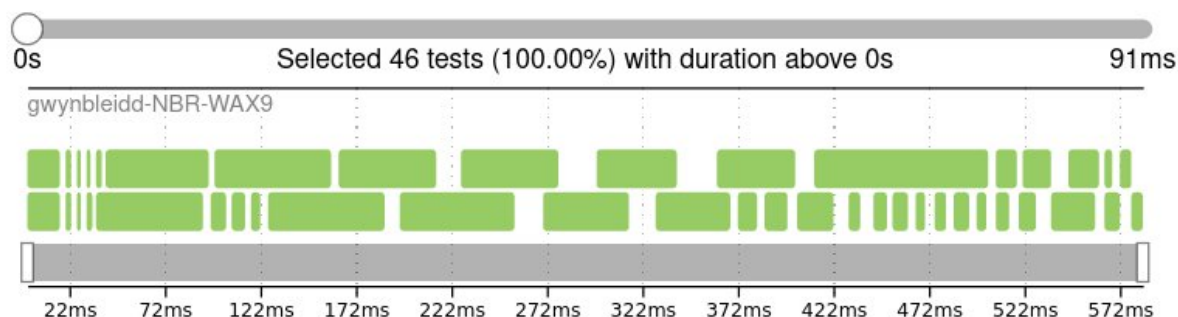


Рисунок 25 — Хронология запуска тестов

Статистика запуска тестов из отчета allure приведена на рисунке 26.

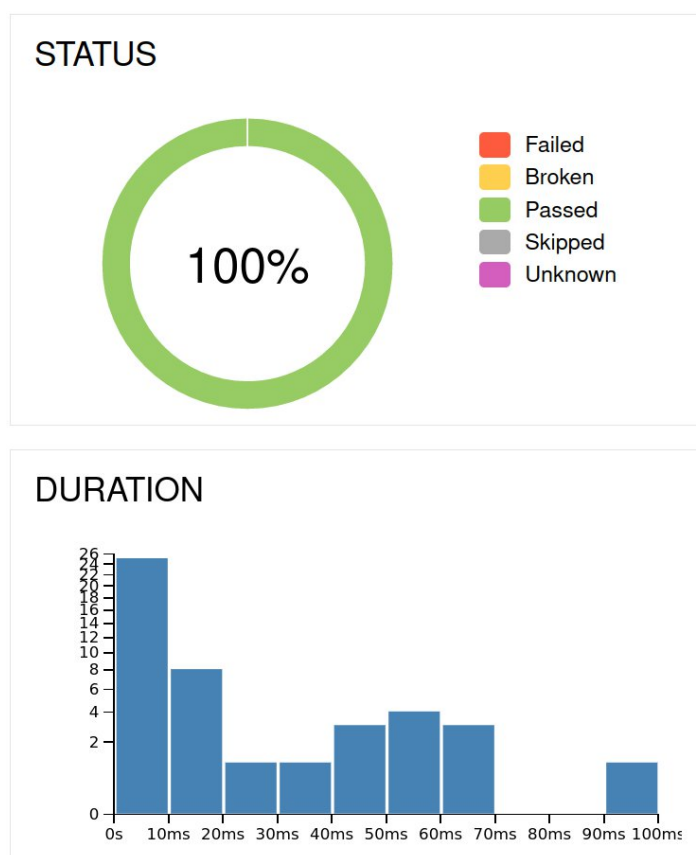


Рисунок 26 — Статистика запуска тестов

Все тесты завершились успешно, о чем свидетельствует приведенная статистика.

3.6 Нагрузочное тестирование

Для упрощения поддержки тестов реализовать нагрузочное тестирование было решено так же с помощью Python. Наиболее популярной библиотекой для нагрузочного тестирования с помощью Python является locust. Locust предоставляет веб-интерфейс для настройки нагрузки и просмотра отчетов.

Пользователь значительно чаще считывает данные из БД, чем пишет в нее, кроме того значительное время может занимать синтез устройств и преобразование временных диаграмм при проверке заданий на программирование.

По этим причинам в ходе нагрузочного тестирования были использованы запросы на чтение к БД, отправленные через основной микросервис (чтобы учесть задержку при проксировании запроса), а так же запросы на синтез устройств и преобразование временных диаграмм непосредственно к соответствующим микросервисам.

Конфигурация оборудования, на котором проводится нагрузочное тестирование:

- ОС: Ubuntu 20.04 focal;
- Ядро: x86_64 Linux 5.15.0-60-generic;
- Процессор: Intel Core i3-10110U @ 4x 4,1 ГГц;
- ОЗУ: 4229 МБ / 7691 МБ.

Так как конфигурация оборудования уступает требуемой в ТЗ, для нагрузочного тестирования была взята максимальная нагрузка в 100 пользователей.

Так как синтез устройства и преобразование временных диаграмм происходят только при проверке задания и являются довольно сложными операциями, во время выполнения которых в пользовательском интерфейсе образовательного портала может быть отображен индикатор загрузки, среднее время выполнения для них было ограничено относительно большим значением в 1000 мс.

Так как считывание данных из БД происходит каждый раз, когда в веб-приложении образовательного необходимо отрисовать список заданий или визуализировать статистику, допустимое среднее время ответа для запросов к БД было установлено равным 300 мс, а 95 перцентиль — 500 мс.

Результаты нагрузочного тестирования преобразователя временных диаграмм приведены в таблице 10.

Таблица 10 — Результаты нагрузочного тестирования преобразователя временных диаграмм

Статистика запросов							
Запросы	Ошибки	Среднее (мс)	Мин. (мс)	Макс. (мс)	Сред. размер (байт)	RPS	Ошибки / с
18458	0	261	3	706	493	279.9	0.0
Статистика ответов							
50%ile (мс)	60%ile (мс)	70%ile (мс)	80%ile (мс)	90%ile (мс)	95%ile (мс)	99%ile (мс)	100%ile (мс)
260	300	350	400	450	470	520	710

Изменения частоты запросов, времени ответа и количества пользователей показаны на рисунках 27-28.



Рисунок 27 — Результаты нагрузочного тестирования преобразователя временных диаграмм

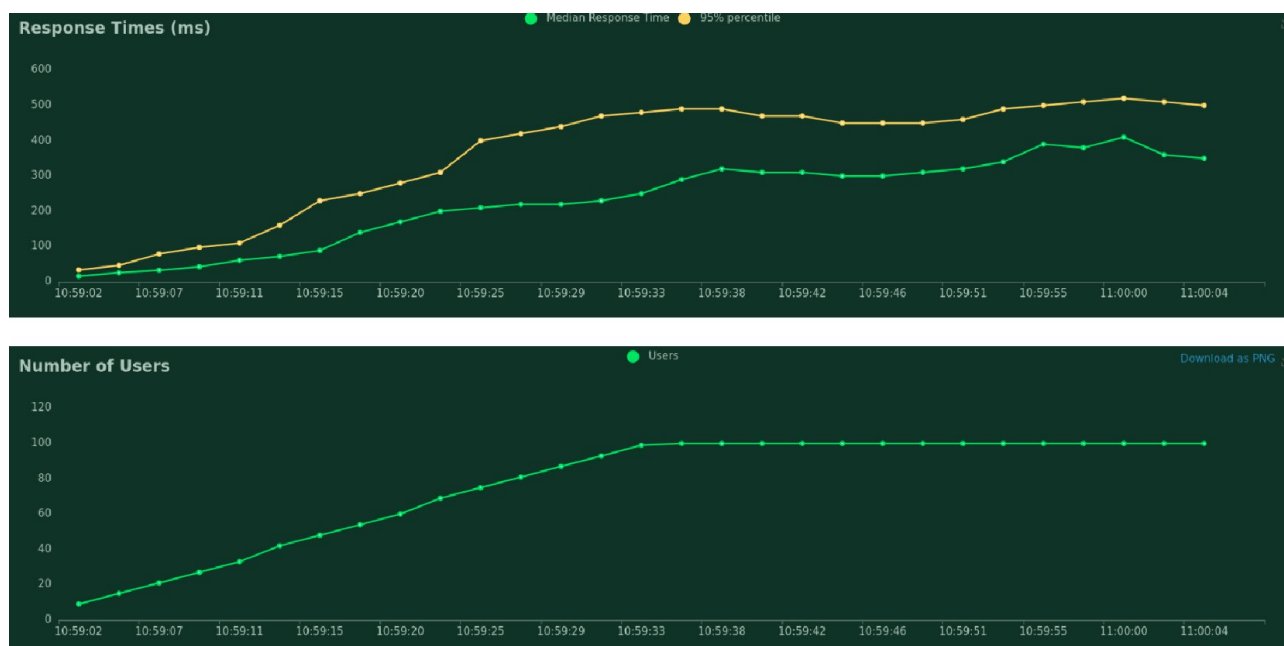


Рисунок 28 — Результаты нагрузочного тестирования преобразователя временных диаграмм

Результаты нагрузочного тестирования синтезатора приведены в таблице 11.

Таблица 11 — Результаты нагрузочного тестирования синтезатора

Статистика запросов							
Запросы	Ошибки	Среднее (мс)	Мин. (мс)	Макс. (мс)	Сред. размер (байт)	RPS	Ошибки / с
12936	0	525	13	1339	3050	152.8	0.0
Статистика ответов							
50%ile (мс)	60%ile (мс)	70%ile (мс)	80%ile (мс)	90%ile (мс)	95%ile (мс)	99%ile (мс)	100%ile (мс)
580	610	640	680	770	870	950	1300

Изменения частоты запросов, времени ответа и количества пользователей показаны на рисунке 29.

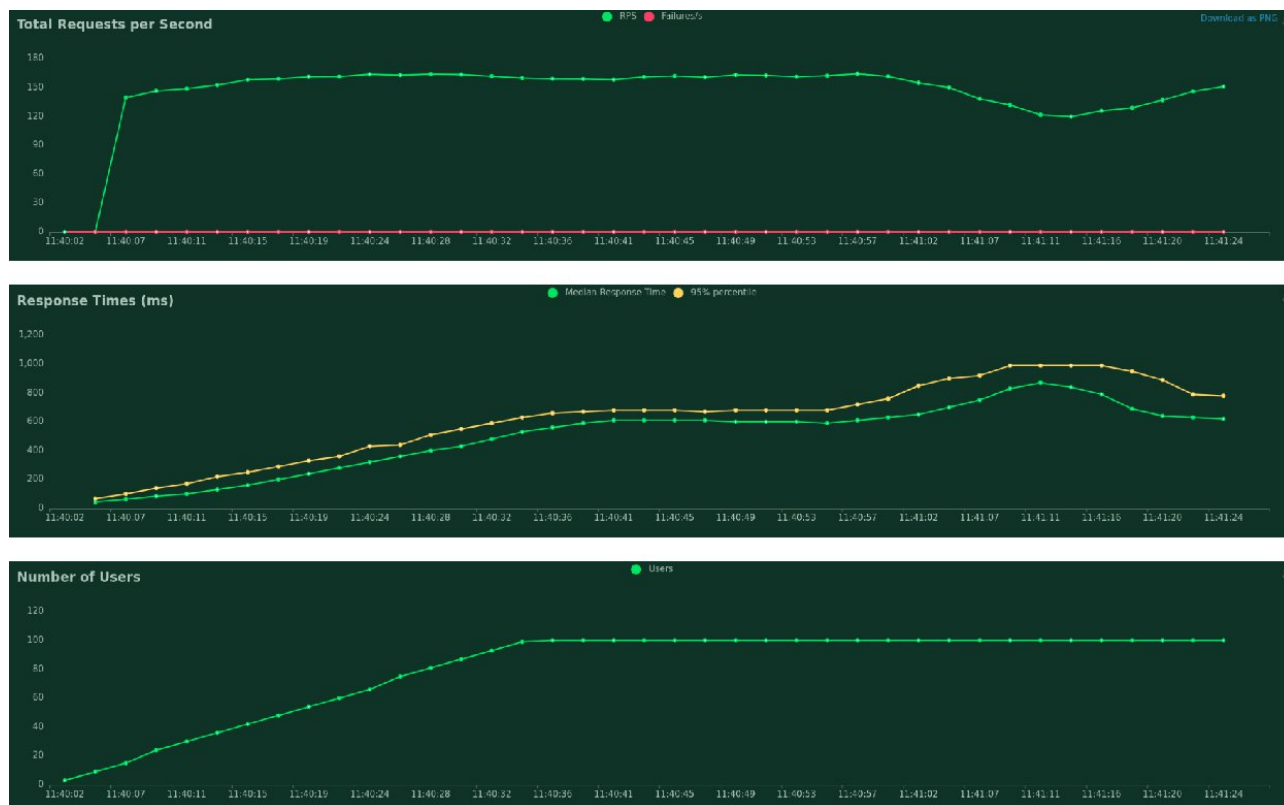


Рисунок 29 — Результаты нагрузочного тестирования синтезатора

Результаты нагрузочного тестирования обращения к БД через слой бизнес-логики приведены в таблице 12.

Таблица 12 — Результаты нагрузочного тестирования обращения к БД через слой бизнес-логики

Статистика запросов								
Маршрут	Запросы	Ошибки	Среднее (мс)	Мин. (мс)	Макс. (мс)	Сред. размер (байт)	RPS	Ошибки / с
/levels	6743	0	85	3	676	506	111.8	0.0
/stats	20486	0	161	2	1151	331	339.5	0.0
Итого	27229	0	142	2	1151	374	451.3	0.0

Продолжение таблицы 12

Статистика ответов								
Маршрут	50%ile (мс)	60%ile (мс)	70%ile (мс)	80%ile (мс)	90%ile (мс)	95%ile (мс)	99%ile (мс)	100%ile (мс)
/levels	78	94	110	130	160	180	220	680
/stats	110	140	190	270	370	460	630	1200
Итого	100	130	160	210	330	430	600	1200

Изменения частоты запросов, времени ответа и количества пользователей показаны на рисунке 30.

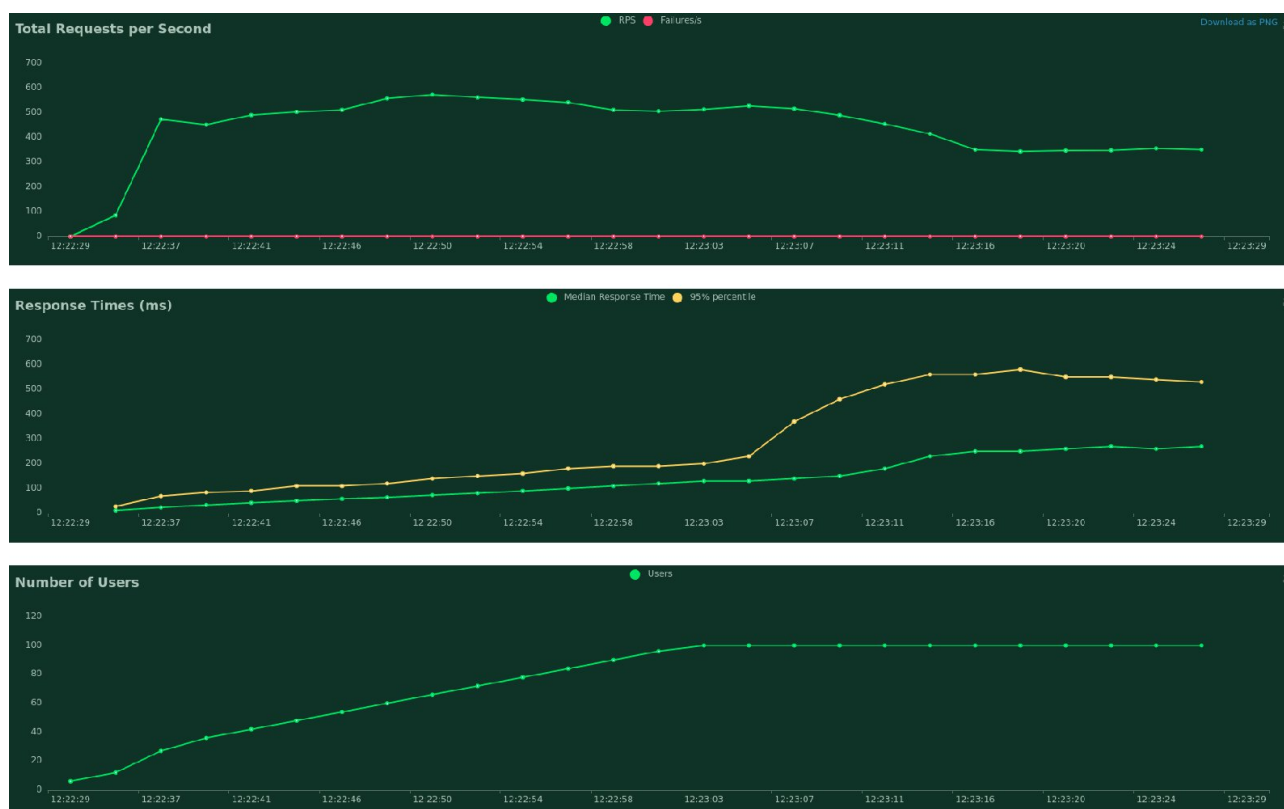


Рисунок 30 — Результаты нагрузочного тестирования обращения к БД через слой бизнес-логики

Тестируемые компоненты в ходе тестирования работали корректно, полученные задержки находятся в допустимых пределах.

3.7 Выводы

В рамках технологической части была разработана технология тестирования. Были проведены автономные и комплексные функциональные тесты на основе подхода черного ящика, а также, нагрузочное тестирование. Функциональные тесты прошли успешно, показатели, полученные в ходе нагрузочного тестирования, находятся в допустимых пределах.

ЗАКЛЮЧЕНИЕ

В рамках выполнения выпускной квалификационной работы бакалавра «Программная подсистема тестирования знаний языков описания аппаратуры» заданная подсистема была спроектирована и реализована. Работа была проведена в несколько этапов.

В исследовательской части работы проведен анализ существующих систем тестирования знаний с целью уточнения функциональных требований и определения вариантов использования разработанной подсистемы.

В конструкторской части работы спроектирована программная подсистема тестирования знаний языков описания аппаратуры, позволяющая управлять учебными материалами и заданиями, проводить автоматическую проверку решений пользователей (учащихся). Подсистема предназначена для интеграции в информационную систему образовательного портала.

В технологической части была разработана технология тестирования разработанной подсистемы, проведено ее функциональное и нагрузочное тестирование. Результаты тестирования позволили заключить, что система работает корректно.

Все поставленные задачи выполнены в полном объеме.

В дальнейшем функциональность системы может быть расширена за счет добавления новых методов тестирования знаний и расширения списка графических представлений статистических данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Мовчан И. Н. Роль контроля в обучении студентов вуза // Психология и педагогика: методика и проблемы практического применения. 2008. №1. URL: <https://cyberleninka.ru/article/n/rol-kontrolya-v-obuchenii-studentov-vuza> (дата обращения: 04.10.2022).
2. Ильина Е.А. Технология тестирования знаний студентов с использованием системы Moodle / Е.А. Ильина, Л.Г. Егорова, А.В. Дьяконов // Математическое и программное обеспечение систем в промышленной и социальной сферах . – Магнитогорск, 2011. – С. 166-172.
3. Справочный центр Stepik - Практические задания [Электронный ресурс]. – URL: <https://clck.ru/Nu5Wy> (дата обращения: 20.10.2022)
4. Справочный центр Stepik - Составление заданий с рецензированием [Электронный ресурс]. – URL: <https://clck.ru/32a9FC> (дата обращения: 01.11.2022)
5. Гладких И.Ю., Якушин А.В. Системы автоматизированного тестирования по программированию в образовательном пространстве // Современные проблемы науки и образования. – 2016. – № 3. ; URL: <https://science-education.ru/ru/article/view?id=24719> (дата обращения: 05.11.2022).
6. Иванова Г.С. – Технология программирования: учебник / Г.С. Иванова. – 3-е изд., стер. – М. : КНОРУС, 2016. – 334 с. – (Бакалавриат).
7. C4model [Электронный ресурс]. — URL: <https://c4model.com/> (дата обращения: 01.02.2023).
8. Medium — Полиморфизм с интерфейсами в Golang [Электронный ресурс]. — URL: <https://clck.ru/33Vd5g> (дата обращения: 05.02.2023).
9. ReadTheDocs — pyDigitalWaveTools’s documentation [Электронный ресурс]. — URL: <https://pydigitalwavetools.readthedocs.io/en/latest/> (дата обращения: 09.02.2023).

10. WaveDrom — Hitchhiker's Guide to the WaveDrom [Электронный ресурс]. — URL: <https://wavedrom.com/tutorial.html> (дата обращения: 11.02.2023).
11. Coder Lessons — Тестирование черного и белого ящика [Электронный ресурс]. — URL: <https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/ruchnoe-testirovanie/chernyi-iashchik-protiv-belaia-korobka> (дата обращения: 01.03.2023).
12. Habr — Рейтинг языков программирования 2021 [Электронный ресурс]. — URL: <https://habr.com/ru/post/543346/> (дата обращения: 07.03.2023).
13. Software Testing Help — Top 6 BEST Python Testing Frameworks [Электронный ресурс]. — URL: <https://www.softwaretestinghelp.com/python-testing-frameworks/> (дата обращения: 14.03.2023).
14. Blog Skillfactory — Pytest [Электронный ресурс]. — URL: <https://blog.skillfactory.ru/glossary/pytest/> (дата обращения: 21.03.2023).
15. Allure Framework — Docs [Электронный ресурс]. — URL: <https://docs.qameta.io/allure/> (дата обращения: 28.03.2023).

Приложение А
Техническое задание
Листов 10

Приложение Б

Руководство системного программиста

Листов 16

Приложение В

Графический материал

Листов 7

Приложение Г

Исходный код микросервиса анализа решений

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
    "os/exec"
    "runtime"

    "github.com/gorilla/mux"
)

// позволяет считать тип проверяемого задания из тела http-запроса
type TypeSelector struct {
    Type string `json:"type"`
}

// описывает тело запроса на проверку теста с одним вариантом ответа
type SingleChoiceTestRequest struct {
    Type string `json:"type"`
    Data struct {
        UserAnswerID int `json:"user_answer_id"`
        Task struct {
            CorrectAnswerID int `json:"correct_answer_id"`
            Answers []struct {
                Text string `json:"text"`
                Hint string `json:"hint"`
            } `json:"answers"`
        } `json:"task"`
    } `json:"data"`
}

// описывает тело запроса на проверку теста с несколькими вариантами ответа
type MultiChoiceTestRequest struct {
    Type string `json:"type"`
    Data struct {
        UserAnswers []bool `json:"user_answers"`
    }
}
```

```

        Task    struct {
            CorrectAnswers []bool `json:"correct_answers"`
        } `json:"task"`
    } `json:"data"`
}

// описывает поведение цифрового сигнала во времени
type WavedromSignal struct {
    Name string `json:"name"`
    Wave string `json:"wave"`
    Data []string `json:"data"`
}

// описывает тело запроса на проверку задания на написание Verilog-кода
// (сверяться будут временные диаграммы)
type CodeRequest struct {
    Type string `json:"type"`
    Data struct {
        UserSignals []WavedromSignal `json:"user_signals"`
        CorrectSignals []WavedromSignal `json:"correct_signals"`
    } `json:"data"`
}

// === форматы ответов ===

type SingleChoiceTestResult struct {
    Hint string `json:"hint"`
}

type MultiChoiceTestResult struct {
    FalsePositive bool `json:"false_positive"`
    FalseNegative bool `json:"false_negative"`
}

type CodeResult struct {
    MissingSignals []string `json:"missing_signals"`
    MismatchingSignals []string `json:"mismatching_signals"`
}

```



```

// формат полного ответа
type ResponseFrame struct {
    StatusStr string    `json:"status_str"`
    StatusCode int       `json:"status_code"`
    Message string    `json:"message"`
    IsCorrect bool      `json:"is_correct"`
    // SingleChoiceTestResult, MultiChoiceTestResult или CodeResult
    Data interface{} `json:"data"`
}

// интерфейс для проверки заданий
type ICheckable interface {
    Check() (bool, interface{})
}

// метод для проверки задания с одним вариантом ответа
func (v SingleChoiceTestRequest) Check() (bool, interface{}) {
    var fl bool
    fl = (v.Data.UserAnswerID == v.Data.Task.CorrectAnswerID)
    var res SingleChoiceTestResult
    res.Hint = v.Data.Task.Answers[v.Data.UserAnswerID].Hint
    return fl, res
}

// метод для проверки задания с несколькими вариантами ответа
func (v MultiChoiceTestRequest) Check() (bool, interface{}) {
    var fl, false_positive, false_negative bool
    if len(v.Data.UserAnswers) != len(v.Data.Task.CorrectAnswers) {
        panic("Answers arrays size mismatch")
    }

    fl = true
    false_positive = false
    false_negative = false
    for i, _ := range v.Data.UserAnswers {
        if !v.Data.UserAnswers[i] && v.Data.Task.CorrectAnswers[i] {
            fl = false
            false_negative = true
        } else if v.Data.UserAnswers[i] && !v.Data.Task.CorrectAnswers[i] {

```

```

        fl = false
        false_positive = true
    }
}

var res MultiChoiceTestResult
res.FalsePositive = false_positive
res.FalseNegative = false_negative
return fl, res
}

// метод для проверки задания на написание Verilog-кода (сверки временных диаграмм)
func (v CodeRequest) Check() (bool, interface{}) {
    var fl bool
    var res CodeResult
    fl = true

    // для каждого сигнала из эталонных
    for _, signal_correct := range v.Data.CorrectSignals {
        var entry_fl bool
        entry_fl = false
        var signal_user_buf WavedromSignal
        // найти соответствующий сигнал в врем. диаграмме пользователя
        for _, signal_user := range v.Data.UserSignals {
            if signal_correct.Name == signal_user.Name {
                entry_fl = true
                signal_user_buf = signal_user
            }
        }
        if entry_fl { // если найден, то проверить соответствие значений
            var equality_fl bool
            equality_fl = true
            if signal_correct.Wave != signal_user_buf.Wave {
                equality_fl = false
            }
            if len(signal_correct.Data) != len(signal_user_buf.Data) {
                equality_fl = false
            } else {
                for i, _ := range signal_correct.Data {
                    if signal_correct.Data[i] != signal_user_buf.Data[i] {

```

```

                                equality_fl = false
                                break
                            }
                        }
                    }
                if !equality_fl {
                    // если несоответствуют – добавить в список несоответствий
                    res.MismatchingSignals = append(res.MismatchingSignals,
                        signal_correct.Name)
                    fl = false
                }
            } else {
                // если сигнал не найден – добавить в список ненайденных
                res.MissingSignals = append(res.MissingSignals, signal_correct.Name)
                fl = false
            }
        }
    }

    return fl, res
}

// обработка http-запросов
func handler(w http.ResponseWriter, req *http.Request) {
    var response ResponseFrame

    // обработка ошибок
    defer func() {
        if panicInfo := recover(); panicInfo != nil {
            var response ResponseFrame
            response.StatusStr = "error"
            response.StatusCode = 400
            response.Message = fmt.Sprintf("Top-level panic: %v", panicInfo)
            w.WriteHeader(response.StatusCode)
            json.NewEncoder(w).Encode(response)
        }
    }()

    reqBody, _ := ioutil.ReadAll(req.Body) // считать тело запроса
    var type_selector TypeSelector

```

```

err := json.Unmarshal(reqBody, &type_selector) // определение типа задания

if err != nil {
    defer func() {
        if r := recover(); r != nil {
            response.StatusStr = "error"
            response.StatusCode = 400
            response.Message = err.Error()
            w.WriteHeader(response.StatusCode)
            json.NewEncoder(w).Encode(response)
        }
    }()
    panic("request-JSON parsing error")
}

// разбор тела запроса в зависимости от типа задания
var data interface{}
if type_selector.Type == "singlechoice_test" {
    data = &SingleChoiceTestRequest{}
} else if type_selector.Type == "multichoice_test" {
    data = &MultiChoiceTestRequest{}
} else if type_selector.Type == "program" {
    data = &CodeRequest{}
} else {
    panic("Unknown task type")
}

err = json.Unmarshal(reqBody, data)

if err != nil {
    defer func() {
        if r := recover(); r != nil {
            response.StatusStr = "error"
            response.StatusCode = 400
            response.Message = err.Error()
            w.WriteHeader(response.StatusCode)
            json.NewEncoder(w).Encode(response)
        }
    }()
    panic("data-JSON parsing error")
}

```

```

    }

    // проверить задание
    response.IsCorrect, response.Data = data.(ICheckable).Check()

    response.StatusStr = "ok"
    response.StatusCode = 200
    response.Message = "checked"
    w.WriteHeader(response.StatusCode)
    json.NewEncoder(w).Encode(response)
}

// инициализация сервера
func main() {
    if runtime.GOOS == "windows" {
        fmt.Println("Can't Execute this on a windows machine")
    } else {
        fmt.Println("Server start")

        r := mux.NewRouter().StrictSlash(true)
        r.HandleFunc("/check", handler).Methods("POST")
        http.ListenAndServe(":8083", r)

        fmt.Println("Server stop")
    }
}

```

Приложение Д

Исходные коды на языке Verilog

Листинг Д.1 — исходный код описания устройства

```
module half_adder(
    output S,C,
    input A,B
);
xor(S,A,B);
and(C,A,B);
endmodule

module full_adder(
    output S,Cout,
    input A,B,Cin
);
wire s1,c1,c2;
half_adder HA1(s1,c1,A,B);
half_adder HA2(S,c2,s1,Cin);
or OG1(Cout,c1,c2);

endmodule

module ripple_adder_4bit(
    output [3:0] Sum,
    output Cout,
    input [3:0] A,B,
    input Cin
);
wire c1,c2,c3;
full_adder FA1(Sum[0],c1,A[0],B[0],Cin),
FA2(Sum[1],c2,A[1],B[1],c1),
FA3(Sum[2],c3,A[2],B[2],c2),
FA4(Sum[3],Cout,A[3],B[3],c3);

endmodule
```

Листинг Д.2 — исходный код тестирующей программы

```
module adder_tb;
// Inputs
reg [3:0] A;
reg [3:0] B;
reg Cin;
// Outputs
wire [3:0] Sum;
wire Cout;
// Instantiate the Unit Under Test (UUT)
ripple_adder_4bit uut (
.Sum(Sum),
.Cout(Cout),
.A(A),
.B(B),
.Cin(Cin)
);
initial begin
// Initialize Inputs
A = 0;
B = 0;
Cin = 0;
// Wait 100 ns for global reset to finish
#100;
// Add stimulus here
A=4'b0001;B=4'b0000;Cin=1'b0;
#10 A=4'b1010;B=4'b0011;Cin=1'b0;
#10 A=4'b1101;B=4'b1010;Cin=1'b1;
end
initial begin
$dumpfile("adder.vcd");
$dumpvars;
end
endmodule
```

Приложение Е

Временная диаграмма в формате VCD

Листинг Е.1 — пример временной диаграммы в формате VCD

```
$date
    Sun Aug 21 20:24:04 2022
$end
$version
    Icarus Verilog
$end
$timescale
    1s
$end
$scope module adder_tb $end
$var wire 4 ! Sum [3:0] $end
$var wire 1 " Cout $end
$var reg 4 # A [3:0] $end
$var reg 4 $ B [3:0] $end
$var reg 1 % Cin $end
$scope module uut $end
$var wire 4 & A [3:0] $end
$var wire 4 ' B [3:0] $end
$var wire 1 % Cin $end
$var wire 1 ( c3 $end
$var wire 1 ) c2 $end
$var wire 1 * c1 $end
$var wire 4 + Sum [3:0] $end
$var wire 1 " Cout $end
$scope module FA1 $end
$var wire 1 , A $end
$var wire 1 - B $end
$var wire 1 % Cin $end
$var wire 1 * Cout $end
$var wire 1 . s1 $end
$var wire 1 / c2 $end
$var wire 1 0 c1 $end
$var wire 1 1 S $end
$scope module HA1 $end
$var wire 1 , A $end
$var wire 1 - B $end
$var wire 1 0 C $end
```



```

$var wire 1 . S $end
$upscope $end
$scope module HA2 $end
$var wire 1 . A $end
$var wire 1 % B $end
$var wire 1 / C $end
$var wire 1 1 S $end
$upscope $end
$upscope $end
$scope module FA2 $end
$var wire 1 2 A $end
$var wire 1 3 B $end
$var wire 1 * Cin $end
$var wire 1 ) Cout $end
$var wire 1 4 s1 $end
$var wire 1 5 c2 $end
$var wire 1 6 c1 $end
$var wire 1 7 S $end
$scope module HA1 $end
$var wire 1 2 A $end
$var wire 1 3 B $end
$var wire 1 6 C $end
$var wire 1 4 S $end
$upscope $end
$scope module HA2 $end
$var wire 1 4 A $end
$var wire 1 * B $end
$var wire 1 5 C $end
$var wire 1 7 S $end
$upscope $end
$upscope $end
$scope module FA3 $end
$var wire 1 8 A $end
$var wire 1 9 B $end
$var wire 1 ) Cin $end
$var wire 1 ( Cout $end
$var wire 1 : s1 $end
$var wire 1 ; c2 $end
$var wire 1 < c1 $end
$var wire 1 = S $end
$scope module HA1 $end

```

```

$var wire 1 8 A $end
$var wire 1 9 B $end
$var wire 1 < C $end
$var wire 1 : S $end
$upscope $end
$scope module HA2 $end
$var wire 1 : A $end
$var wire 1 ) B $end
$var wire 1 ; C $end
$var wire 1 = S $end
$upscope $end
$upscope $end
$scope module FA4 $end
$var wire 1 > A $end
$var wire 1 ? B $end
$var wire 1 ( Cin $end
$var wire 1 " Cout $end
$var wire 1 @ s1 $end
$var wire 1 A c2 $end
$var wire 1 B c1 $end
$var wire 1 C S $end
$scope module HA1 $end
$var wire 1 > A $end
$var wire 1 ? B $end
$var wire 1 B C $end
$var wire 1 @ S $end
$upscope $end
$scope module HA2 $end
$var wire 1 @ A $end
$var wire 1 ( B $end
$var wire 1 A C $end
$var wire 1 C S $end
$upscope $end
$upscope $end
$upscope $end
$upscope $end
$enddefinitions $end
#0
$dumpvars
0C
0B

```

0A
0@
0?
0>
0=
0<
0;
0:
09
08
07
06
05
04
03
02
01
00
0/
0.
0-
0,
b0 +
0*
0)
0(
b0 '
b0 &
0%
b0 \$
b0 #
0"
b0 !
\$end
#100
b1 !
b1 +
11
1.
1,
b1 #

b1 &
#110
1=
1)
b1101!
b1101 +
1C
16
1@
1-
13
0,
12
1>
b11 \$
b11 '
b1010 #
b1010 &
#120
1(
1C
1"
15
1)
0=
1;
1*
0@
1B
14
06
1:
b1000!
b1000 +
01
1/
0-
1?
1,
02
18

1%

b1010 \$

b1010 '

b1101 #

b1101 &