

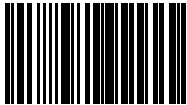
Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS

Разработка автоматизированного трекера для настольных игр

Обучающийся / Student Конаныхина Антонина Александровна
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники
Группа/Group Р34102
Направление подготовки/ Subject area 09.03.04 Программная инженерия
Образовательная программа / Educational program Системное и прикладное программное обеспечение 2020
Язык реализации ОП / Language of the educational program Русский
Квалификация/ Degree level Бакалавр
Руководитель ВКР/ Thesis supervisor Пинкевич Василий Юрьевич, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Обучающийся/Student

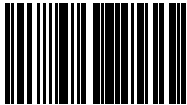
Документ подписан	
Конаныхина Антонина Александровна	
21.05.2024	

(эл. подпись/ signature)

Конаныхина
Антонина
Александровна

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Пинкевич Василий Юрьевич	
20.05.2024	

(эл. подпись/ signature)

Пинкевич
Василий
Юрьевич

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Конаныхина Антонина Александровна
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники
Группа/Group P34102
Направление подготовки/ Subject area 09.03.04 Программная инженерия
Образовательная программа / Educational program Системное и прикладное программное обеспечение 2020
Язык реализации ОП / Language of the educational program Русский
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Разработка автоматизированного трекера для настольных игр
Руководитель ВКР/ Thesis supervisor Пинкевич Василий Юрьевич, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Характеристика темы ВКР / Description of thesis subject (topic)

Тема в области фундаментальных исследований / Subject of fundamental research: нет / not

Тема в области прикладных исследований / Subject of applied research: да / yes

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Техническое задание:

- 1) Провести анализ рынка, найти существующие решения, или проанализировать почему они отсутствуют.
- 2) Разработка аппаратного обеспечения, принципиальной схемы, схемы печатной платы, изготовление печатной платы.
- 3) Разработка программного обеспечения. Разработка программного обеспечения для микроконтроллера, разработка программного обеспечения для персональных компьютеров.
- 4) Тестирование и испытание устройства. Сбор технических характеристик, энергозатрат, времени отклика.

Исходные данные к работе:

- 1) Luo X. Design of Portable ECG Device Based on STM32 //2016 6th International Conference on Machinery, Materials, Environment, Biotechnology and Computer. – Atlantis Press, 2016. – С. 386-389.
- 2) Tan Z. et al. Design of Smart Card Chip Reader Based on STM32 //Journal of Networking and Telecommunications. – 2020. – Т. 2. – №. 1. – С. 7-11.
- 3) Составное устройство USB на STM32. [Электронный ресурс]. URL:

<https://habr.com/ru/articles/532038/>

Цели и задачи работы:

Цель: улучшение качества игрового процесса в настольных играх за счет автоматизации

Задачи:

- 1) Анализ существующих решений или причины их отсутствия
- 2) Исследование работы и способов разводки принципиальных электрических схем на базе герконов
- 3) Разработка и изготовление платы расширения для микроконтроллера STM32 Nucleo
- 4) Написание ПО для конфигурации и настройки трекера и UI для отображения обработанных данных
- 5) Тестирование разработанного ПО на изготовленной плате
- 6) Оформление документации

Перечень подлежащих разработке вопросов:

- 1) Поиск оптимального способа подключения матрицы герконов.
- 2) Оценка физических параметров устройства: энергопотребление, скорость отклика.
- 3) Оценка субъективных параметров: удобство использования.

Рекомендуемые материалы и пособия для выполнения работы:

- 1) А.О. Ключев, Д.Р. Ковязина, П.В. Кустарев, А.Е. Платунов. Аппаратные и программные средства встраиваемых систем - Санкт-Петербург: Университет ИТМО, 2010. - 290 с. - 100 экз.
- 2) А.О. Ключев, Д.Р. Ковязина, Е.В. Петров, А.Е. Платунов. Интерфейсы периферийных устройств. - Санкт-Петербург: СПбГУ ИТМО, 2010. - 294 с. - 100 экз.

Форма представления материалов ВКР / Format(s) of thesis materials:

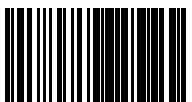
Презентация, пояснительная записка

Дата выдачи задания / Assignment issued on: 31.10.2023

Срок представления готовой ВКР / Deadline for final edition of the thesis 30.05.2024

СОГЛАСОВАНО / AGREED:

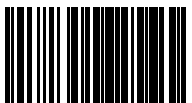
Руководитель ВКР/
Thesis supervisor

Документ подписан	
Пинкевич Василий Юрьевич	
02.05.2024	

(эл. подпись)

Пинкевич
Василий
Юрьевич

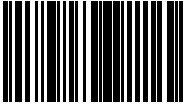
Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан	
Конаныхина Антонина Александровна	
14.05.2024	

(эл. подпись)

Конаныхина
Антонина
Александровна

Руководитель ОП/ Head
of educational program

Документ подписан	
Дергачев Андрей Михайлович	
22.05.2024	

(эл. подпись)

Дергачев
Андрей
Михайлович

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Конаныхина Антонина Александровна
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники
Группа/Group Р34102
Направление подготовки/ Subject area 09.03.04 Программная инженерия
Образовательная программа / Educational program Системное и прикладное программное обеспечение 2020
Язык реализации ОП / Language of the educational program Русский
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Разработка автоматизированного трекера для настольных игр
Руководитель ВКР/ Thesis supervisor Пинкевич Василий Юрьевич, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Улучшение качества игрового процесса в настольных играх за счет автоматизации

Задачи, решаемые в ВКР / Research tasks

1) Исследование предметной области, поиск аналогов разрабатываемого прототипа устройства или поиск причин их отсутствия. 2) Сравнительный анализ различных схем подключения матрицы герконов к микроконтроллеру. 3) Разработка принципиальной электрической схемы прототипа трекера и изготовление тестового стенда. 3) Разработка программного обеспечения прототипа трекера. 3) Тестирование готового прототипа, определение физических и пользовательских характеристик и анализ полученных результатов.

Краткая характеристика полученных результатов / Short summary of results/findings

Разработан и протестирован тестовый стенд прототипа трекера на макетной плате. Разработано программное обеспечение для тестового стенда и графический интерфейс для интерпретации и отображения собранной с датчиков информации.

Обучающийся/Student

Документ подписан	
Конаныхина Антонина	
Александровна	

Конаныхина
Антонина

21.05.2024	
------------	--

(эл. подпись/ signature)

Александровна

(Фамилия И.О./ name and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Пинкевич Василий Юрьевич	
20.05.2024	

(эл. подпись/ signature)

Пинкевич
Василий
Юрьевич

(Фамилия И.О./ name and surname)

ОГЛАВЛЕНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ.....	4
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ.....	5
ВВЕДЕНИЕ.....	7
1 Введение в предметную область	9
1.1 Внедрение автоматизации в настольные игры	9
1.2 Причины малого количества случаев автоматизации.....	9
1.3 Обзор аналогов разрабатываемого устройства.....	11
1.4 Постановка задачи	14
2 Разработка аппаратного обеспечения трекера	16
2.1 Проблемы выбора организации и подключения герконов	16
2.2 Вычислительная часть трекера.....	21
2.3 Принципиальная электрическая схема	22
2.4 Заключение по разработке аппаратной части	23
3 Разработка программного обеспечения для трекера.....	25
3.1 Программное обеспечение для микроконтроллера STM32.....	25
3.1.1 Конфигурация устройств	25
3.1.2 Разработка базовой логики программы	28
3.1.3 Разработка протокола для взаимодействия программ	32
3.2. Графический интерфейс	39
3.2.1 Создание страниц UI	39
3.2.2 Работа с пользовательской конфигурацией	41
3.3. Заключение по разработке программного обеспечения	45

4 Тестирование и определение характеристик	46
4.1 Определение физических характеристик трекера	47
4.2 Определение пользовательских характеристик.....	47
ЗАКЛЮЧЕНИЕ	50
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	52

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

CTS/RTS – Request To Send / Clear To Send

GPIO – general-purpose input/output

HAL – hardware abstraction layer

UART – universal asynchronous receiver/transmitter

UI – user interface

USB – universal serial bus

АЦП – аналого-цифровой преобразователь

НРИ – настольная ролевая игра

САПР – система автоматизированного проектирования

ПО – программное обеспечение

ПК – персональный компьютер

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Геркон (герметизированный магнитоуправляемый контакт) – элемент электрической цепи, функционально представляющий из себя ключ и меняющий состояние цепи путём механического замыкания или размыкания под воздействием управляющего магнитного поля на герметически изолированные от окружающей среды контакт-детали.

HAL (Hardware Abstraction Layer) – библиотека для разработки программ для STM32, предоставляющая высокоуровневые API с высоким уровнем переносимости между платформами.

GPIO (General-purpose input/output) – порт общего назначения, служит для обмена низкоуровневыми цифровыми сигналами с внешними устройствами.

CTS/RTS (Request To Send / Clear To Send) – опциональный механизм управления потоком и синхронизации в UART.

C++ – компилируемый статически типизированный язык программирования общего назначения, поддерживающий такие парадигмы программирования, как объектно-ориентированное, процедурное и обобщённое программирование.

Python – язык программирования общего назначения со строгой динамической типизацией и автоматическим управлением памятью.

RX/TX (Receiver/Transmitter) – обозначение ввода/вывода канала передачи данных.

UART (Universal asynchronous receiver/transmitter) – семейство протоколов приема-передачи. Самый известный представитель – RS-232.

USB (Universal Serial Bus) – последовательный интерфейс подключения периферийных устройств к ПК.

Виртуальный игровой стол – инструмент для организации и проведения настольных ролевых игр, призванный увеличить удобство онлайн игровых сессий и имитировать в виртуальном пространстве реальный игровой процесс как за обычным игровым столом.

Настольная ролевая игра – один из видов настольных игр, в котором участники устно или письменно отыгрывают своих игровых персонажей в соответствии с их особенностями, а успешность любых действий персонажей определяется бросками различных игровых костей.

Пин – контакт для соединения двух элементов электрической цепи.

ВВЕДЕНИЕ

Во время пандемии многие игроки настольных ролевых игр пересмотрели свой подход к организации очных сессий и стали проводить игровые встречи в онлайн формате, где достаточно удобно отслеживать игровые параметры, следить за игровым окружением и контролировать процесс игры за счёт специализированного ПО, такого как виртуальные столы. Подобное ПО предоставляет широкий функционал для комфортного проведения игр, но, даже несмотря на данные удобства, по прошествии времени многие игроки стали возвращаться в очный формат, так как, по аналогии с набравшим популярность видеоконференциями, дистанционные сессии для многих оказались не тем игровым опытом, который люди хотели приобрести, и возвращение к очным сессиям было неизбежно. Тем не менее, потребность в автоматизации некоторых игровых процессов для улучшения опыта погружения в игру и избавления от отвлекающих формальностей осталась. Поэтому в рамках данной ВКР было решено разрабатывать автоматизированный трекер различных параметров и характеристик для настольных ролевых игр.

Целью работы является улучшение качества игрового процесса в настольных ролевых играх за счет автоматизации.

В рамках работы над данной ВКР должны были быть выполнены следующие **задачи**:

- Исследовать предметную область, найти и проанализировать аналоги разрабатываемого прототипа устройства или причины отсутствия аналогов.
- Провести сравнительный анализ различных схем подключения матрицы датчиков герконов к микроконтроллеру.
- Разработать принципиальную электрическую схему прототипа трекера и изготовить тестовый стенд.

- Разработать программное обеспечение прототипа трекера.
- Протестировать готовый прототип, определить его физические и пользовательские характеристики.
- Проанализировать полученные результаты.

1 Введение в предметную область

1.1 Внедрение автоматизации в настольные игры

В 2020–2021 годах в условиях ограничений, связанных с эпидемией COVID-19, игроки в настольные и настольные ролевые игры искали способы организовать свои игровые сессии несмотря на вынужденную самоизоляцию. Так, согласно данным Google Stats, на фоне пандемии по всему миру выросла популярность виртуальных столов, таких как roll20 и Foundry Virtual Tabletop, являющихся универсальным и удобным решением для проведения игровых сессий в онлайн формате. Как и во многих других отраслях, после снятия ограничений люди стали возвращаться к очному формату игровых партий, так как для многих НРИ – это способ отдохнуть от рабочей рутины и провести время в кругу близких и друзей. Дистанционный формат не предоставляет возможность личного физического контакта и вносит ряд других ограничений, влияющих на игровой процесс, таких как неустойчивость связи, технические неполадки и большие время- и ресурсозатраты на подготовку к игре.

1.2 Причины малого количества случаев автоматизации

Основной фактор, влияющий на то, почему в наши дни автоматизация не особо внедрена в настольные игры и НРИ – желание людей проводить больше времени за живым общением без взаимодействия с электронными устройствами. Однако возвращение к очным играм после длительного использования вышеупомянутых виртуальных столов не могло не сопровождаться появлением различных разработок, связанных с автоматизацией игрового процесса.

Базово для НРИ необходим лишь набор игровых костей для внесения вариативности и элемента случайности в игровой сценарий, поэтому зачастую любые разработки и устройства для подобных игр бывают излишни, но всё же они имеют свои преимущества. Так, любые варианты визуализации информации способствуют разгрузке игроков от лишних расчётов и

позволяют направить все ресурсы на воплощение своих задумок, тем самым повышая общую вовлечённость в игру, что в случае виртуальных столов решалось интерактивными игровыми полями.

Из-за доступности для любых игроков и низкого порога вхождения с точки зрения финансов запросы на улучшение игрового процесса за счёт автоматизации чаще всего единичны или не являются массовыми, так как разрабатываемые устройства либо достаточно дорогие, либо требуют особых условий эксплуатации. Тем не менее, спрос на них есть, а многие игроки при организации своих сессий используют гибридные технологии с использованием различных самостоятельно разработанных устройств. В любом случае, автоматизация части рутинных процессов игры, таких как подсчёт параметров и урона игроков, отслеживание различных игровых состояний персонажей, количество оставшихся ячеек заклинаний у игрока и так далее, лишь улучшает пользовательский опыт и способствует комфортному и более глубокому погружению в игровой процесс.

Для устройства, разрабатываемого в рамках данной ВКР, а именно автоматизированного трекера, предъявляются следующие критерии:

- простота изготовления и эксплуатации;
- удобный функционал, не отвлекающий от игрового процесса;
- дешевизна;
- компактность;
- универсальность.

Основная цель любой НРИ – получить удовольствие от времяпровождения, то есть вполне резонно отказаться от варианта полной замены аналоговых трекеров на использование аналогичного ПО на персональном компьютере во время игровых сессий, так как наличие на столе у игроков физической версии трекера освободит их от ненужных отвлекающих действий, сохранит возможность не пользоваться сторонним ПО

и не отвлекаться от процесса игры для уточнения формальностей, а также поможет ведущему проще координировать действия других игроков и более гибко подстраивать ситуацию на игровом поле под постоянно меняющиеся обстоятельства. Современные исследования показывают, что любой тактильный контакт существенно улучшает эмоциональное, когнитивное и физическое состояния людей, что в случае использования данного трекера лишь способствует получению качественных впечатлений от прошедшей сессии.

1.3 Обзор аналогов разрабатываемого устройства

По описанным в предыдущем пункте причинам существует всего несколько вариантов реализации трекеров для настольных ролевых игр.

1) Аналоговый трекер

Классический трекер для НРИ широко представлен различными изделиями из фанеры. Представляет из себя небольшой планшет с выемками для проставления отметок, обычно соответствующим состоянию «1» («есть») или «0» («нет»). Выемки расположены в виде матрицы, каждой линии и столбцу которой ставятся в соответствие названия отслеживаемой характеристики или параметра (Рисунок 1)

Преимущества:

- низкая цена;
- наглядность;
- простота использования;
- компактность.

Недостатки:

- отсутствие универсальности: для каждой игровой системы/игрового движка необходим отдельный трекер;
- отсутствие какой-либо автоматизации;

- отсутствие возможности ведущему игроку видеть прогресс других участников;
- возможность для участников нарушать правила, так как отсутствует контроль со стороны других игроков и ведущего.



Рисунок 1. Пример аналогового трекера

2) Цифровой стол

Данный стол представляет из себя вмонтированный в обычный стол сенсорный экран с большой диагональю (Рисунок 2). С помощью ПО, такого, как виртуальные столы, возможно демонстрировать игрокам всю необходимую информацию, в том числе организовать для каждого трекинг личных параметров и характеристик. Чаще всего данные столы используются для демонстрации игровых карт и окружения, однако можно настроить их и под другие задачи. У ведущего игрока есть возможность отслеживать текущий прогресс каждого из участников и контролировать ситуацию да столом. Массово в продаже данные столы не существуют, так как каждый пользователь подстраивает функционал и внешний вид под индивидуальные требования.

Преимущества:

- существенное повышение вовлечённости участников в игровой процесс;
- удобная демонстрация данных и сбор необходимых игровых параметров.

Недостатки:

- высокая цена на готовые решения;
- высокие требования к помещению и условиям эксплуатации для проведения игр;
- сложность и медленная скорость подготовки игровых сессий;
- необходимость обучения пользователя работе со сторонним ПО.

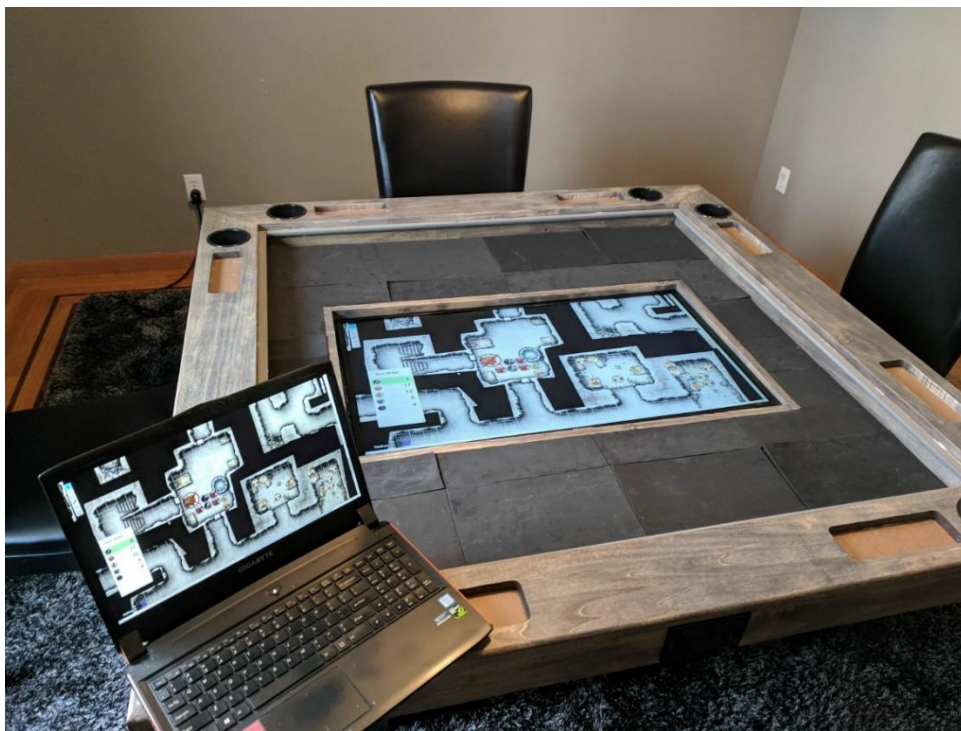


Рисунок 2. Пример цифрового стола

Таким образом, для создания своего устройства следует учитывать требования к стоимости, удобству использования, наглядности и универсальности.

1.4 Постановка задачи

Разрабатываемый автоматизированный трекер вбирает в себя преимущества обоих рассмотренных вариантов реализаций трекеров. Он, как и аналоговый трекер, представляет из себя матрицу с выемками, под которыми расположены датчики герконы. Собранный с датчиков трекера информация о проставленных игроком магнитных метках поступает на микроконтроллер, который в свою очередь передаёт обработанную информацию о расположении меток на ПК ведущего игрока, где, согласно заранее заданной конфигурации для используемой игровой системы, полученные данные обрабатываются и интерпретируются правильным образом. Как итог, ведущий имеет перед глазами текущую информацию обо всех персонажах игроков за столом и способен более гибко подстраивать свой сценарий под происходящее на игровом поле.

Преимущества:

- низкая цена
- удобство использования
- компактность
- универсальность: есть возможность настроить трекер под любую игровую систему
- автоматизация расчётов, как следствие увеличение уровня вовлечённости игроков
- наглядная демонстрация данных для всех игроков
- отсутствие возможности нарушать правила

Недостатки:

- необходимость использования отдельного трекера для каждого игрока
- необходимость изготавливать новую крышку корпуса с выемками под каждую систему.

С помощью двух и более таких трекеров возможно организовать удобную для ведущего игрока систему отслеживания текущих состояний персонажей остальных игроков (Рисунок 3). Все данные, поступившие к ведущему, выводятся посредством графического интерфейса и удобно демонстрируют прогресс каждого игрока.

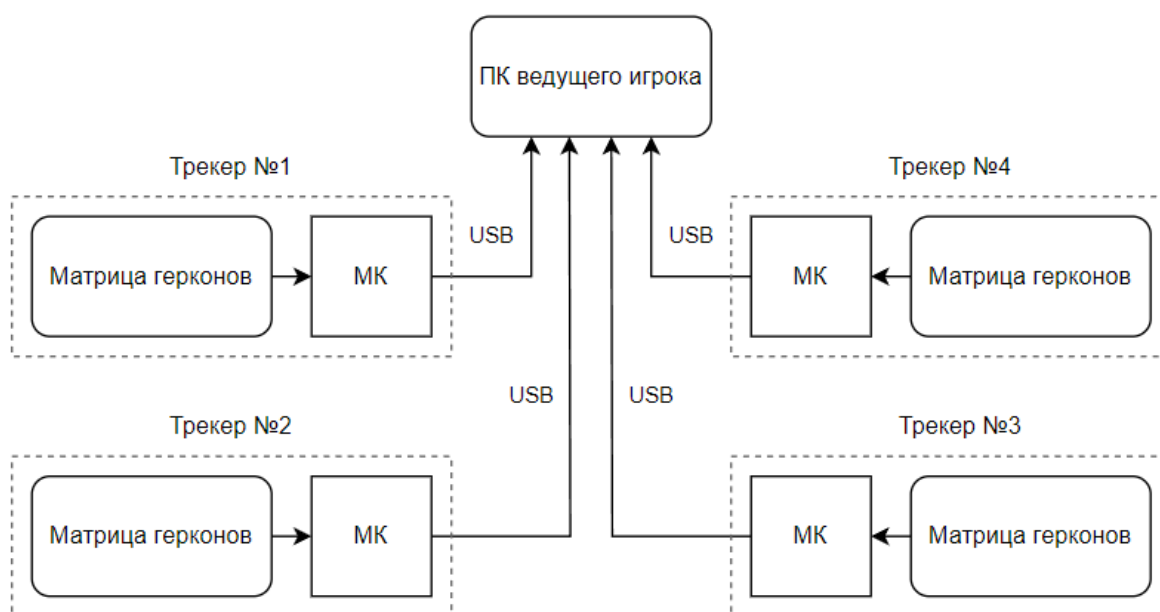


Рисунок 3. Схема работы системы из нескольких трекеров

При разработке прототипа трекера следует учитывать следующие функциональные требования:

1. Система должна собирать данные о наличии магнитных меток на устройстве, об их расположениях и количестве.
2. Система, основываясь на полученном расположении меток, должна находить численные значения характеристик, соответствующих текущему расположению меток. В зависимости от настроек конфигурации численные значения могут рассчитываться исходя из положения меток, их количества или взаимного расположения;
3. Собранная информация должна передаваться устройством на персональный компьютер, подключённый к нему по интерфейсу USB;

4. На персональном компьютере информация должна отображаться пользователю через графический интерфейс;

5. Программное обеспечение должно предоставлять пользователю возможность настройки отображения и интерпретации полученных данных.

2 Разработка аппаратного обеспечения трекера

Аппаратное обеспечение трекера, соответствующее сформулированным требованиям, в первую очередь состоит из двух крупных частей – из **матрицы датчиков герконов**, которые будут определять наличие или отсутствие в конкретной выемке магнитной метки, и **вычислительной части**, которая считывает данные с матрицы герконов и отправляет их на ПК ведущего.

2.1 Проблемы выбора организации и подключения герконов

Первично при разработке возникает проблема построения матрицы герконов, так как от данного выбора организации подключения и построения самой матрицы зависит количество используемых пинов микроконтроллера. Так как герконы по своей сути есть то же самое, что и ключи и кнопки, то можно отталкиваться от известных вариантов подключения данных элементов цепи. Из всех возможных вариантов рассмотрим три наиболее подходящих для данной задачи способа построения подобной матрицы:

1) Прямое подключение каждого датчика

Самый просто способ подключения – это подключить каждый из герконов к отдельному входу микроконтроллера. То есть каждому геркону ставится в соответствие один из GPIO пинов микроконтроллера. Схема подключения для матрицы 3 на 3 представлена ниже (Рисунок 4).

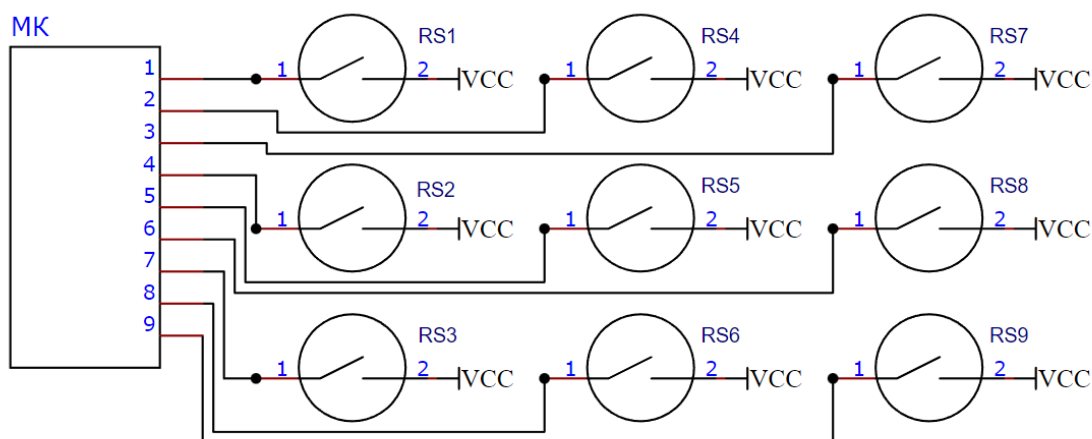


Рисунок 4. Схема прямого подключения датчиков

Преимущества данного способа подключения:

- самый простой в реализации способ;
- удобная организация чтения данных;
- возможно организовать чтения данных по требованию.

Недостатки:

- для матрицы достаточно большого размера необходимо иметь огромное количество свободных цифровых вводов;
- программа для чтения данных и сбора их в матрицу сложнее, чем при других способах;
- с увеличением количества герконов ухудшается масштабируемость матрицы из-за недостатка вводов.

Применение данного метода возможно, когда матрица небольшого размера, а микроконтроллер имеет достаточное количество вводов, равное количеству используемых герконов. Но так как разрабатываемую систему необходимо сделать расширяемой, применение данного метода невозможно.

2) Подключение герконов по столбцам и строкам

Данный способ подключения наиболее распространён на практике. Например, он используется в матричной клавиатуре. Способ устроен

следующим образом: все строки с одного из контактов геркона подключаются к выводам микроконтроллера, а все колонки с другого контакта подключаются ко входам микроконтроллера. Схема подключения для матрицы 3 на 3 представлена ниже (Рисунок 5). Для данной схемы необходимо использовать подтягивающие резисторы внутри микроконтроллера, так как иначе те входы, которые не включены в данный момент, будут подключены к контакту, не подключенному ни к чему. Возможно использование внешних резисторов.

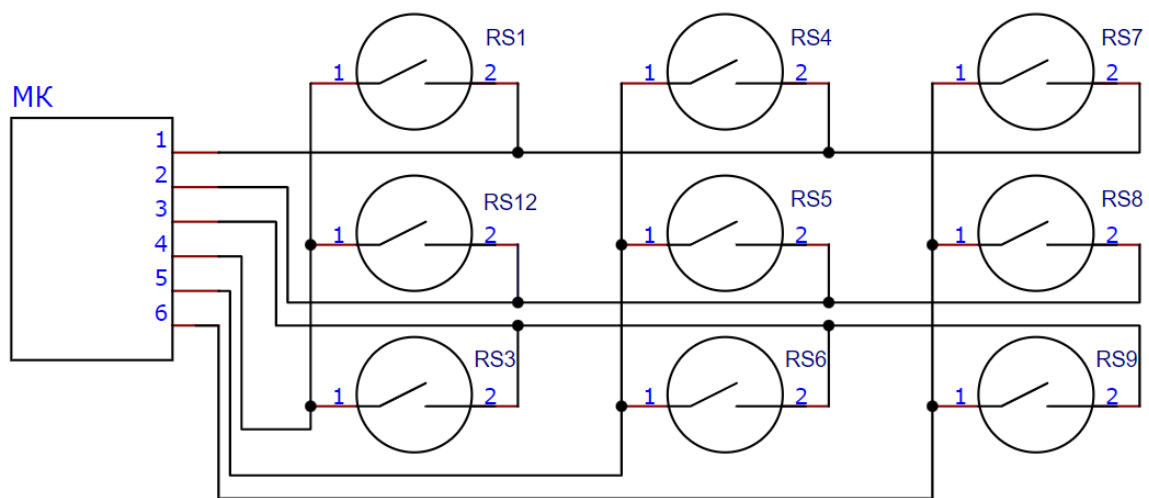


Рисунок 5. Схема подключения по столбцам и строкам

Преимущества:

- по сравнению с прямым подключением необходимо использовать меньше входов-выходов;
- легко написать программу, собирающую данные в матрицу;
- надёжный и проверенный способ подключения.

Недостатки:

- считывание данных происходит дольше, так как необходимо предварительно включить цифровой выход;
- неплохая масштабируемость, но при очень больших размерах матрицы всё ещё потребуется большое количество входов и

выходов. Тем не менее, она сильно лучше, чем при прямом подключении.

- Возникновение фантомных нажатий (подробнее в разделе 4 Тестирование и определение характеристик)

3) Подключение через аналоговый вход с суммированием токов

Данный метод работает за счёт суммирования входящих в узел токов. Одни контакты геркона подключаются к АЦП входу микроконтроллера, а на другие поступает разная сила тока. Так как сумма токов любых замкнутых контактов на выходе всегда будет разной, можно с лёгкостью установить, какие из герконов замкнуты, лишь считав напряжение на АЦП. Для того, чтобы измерить ток, необходимо подключение шунтирующих резисторов. В этом методе можно считывать только линии, так как последующая обработка конечных результатов и по столбцам, и по линиям будет происходить уже на высоком уровне. Схема подключения для матрицы 3 на 3 представлена ниже (Рисунок 6). Номиналы резисторов для данной схемы рассчитываются следующим образом. Берётся базовое сопротивление R_1 , а сопротивления последующих резисторов рассчитывается по формуле:

$$R_n = \frac{R_1}{2^{n-1}}$$

где n – номер резистора. А резисторы R_{10} , R_{11} , R_{12} являются шунтирующими и должны быть как можно меньшего номинала, чтобы исключить помехи. Если использовать большое количество герконов, необходимо усилить напряжение после шунтирующего резистора, так как ток может быть очень мал, а, соответственно, будет мало и напряжение на резисторе. Возможно также использование отдельных микросхем амперметров.

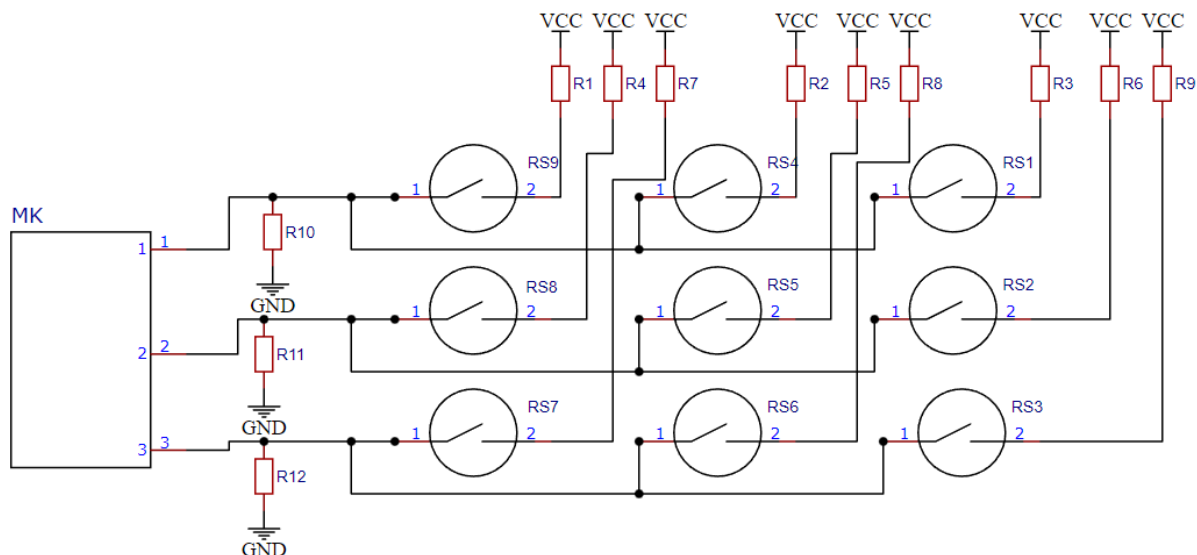


Рисунок 6. Схема подключения, основанная на сумме токов

Преимущества:

- среди всех рассмотренных способов подключения имеет наименьшее необходимое количество вводов.

Недостатки:

- для реализации схемы необходим хороший и многоразрядный АЦП;
- частота опроса датчиков привязана к частоте обновления АЦП;
- плохая масштабируемость из-за того, что при увеличении количества герконов в схему необходимо добавлять усилители, необходима большая разрядность АЦП;
- большая сложность схемы по сравнению с предыдущими способами подключения.

На практике данный способ часто применяется при подключении небольшого количества ключей, но, к сожалению, ввиду того, что платформа, на которой ведётся разработка, имеет низкоразрядный АЦП, а также ввиду большей сложности схемы подключения в целом и плохой масштабируемости, данное решение не будет применяться в рамках проекта.

Для реализации в прототипе трекера в итоге был выбран **второй вариант подключения**, по столбцам и строкам, как оптимальный по всем требуемым для прототипа критериям. На макетной плате была разведена матрица герконов 6 на 6, как минимально подходящая по количеству для задач трекинга.

2.2 Вычислительная часть трекера

Помимо непосредственно матрицы герконов устройство также имеет вычислительную платформу. Данная часть платы может быть представлена любым микроконтроллером, но в прототипе в этом качестве используется плата STM32 NucleoF334 как один из самых распространённых и низких по цене в линейке.

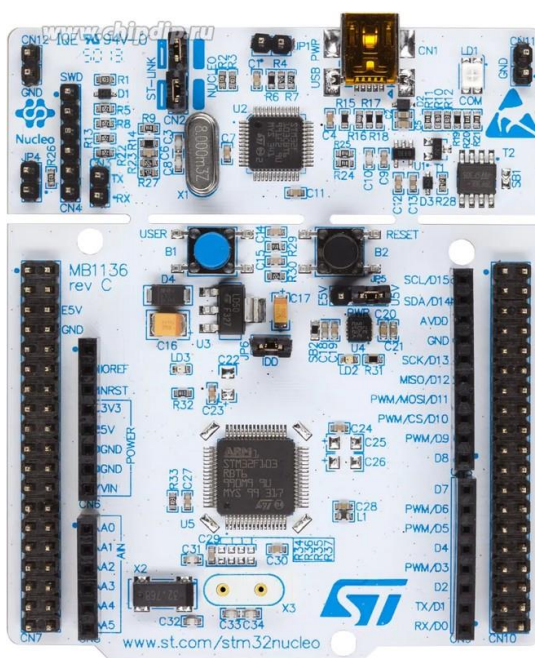


Рисунок 7. Внешний вид STM32 NucleoF334

Данный контроллер удобен тем, что на нем сразу установлен программатор и имеется достаточное количество вводов-выводов под решаемые в рамках проекта задачи. Использование этого устройства значительно упростит процесс разработки.

2.3 Принципиальная электрическая схема

Так как разрабатывается прототип трекера, то было решено развести матрицу герконов размером лишь 6 на 6 датчиков и подключить их по столбцам и строкам. С использованием САПР EasyEDA была спроектирована принципиальная электрическая схема устройства.

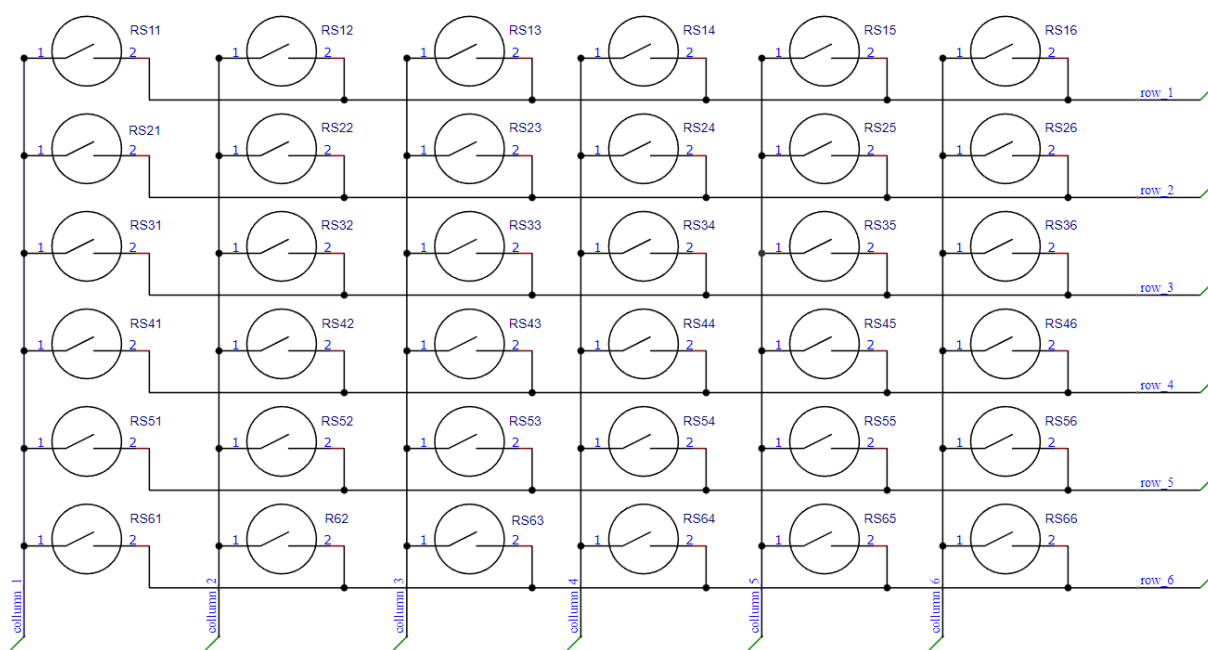


Рисунок 8. Часть схемы с матрицей герконов

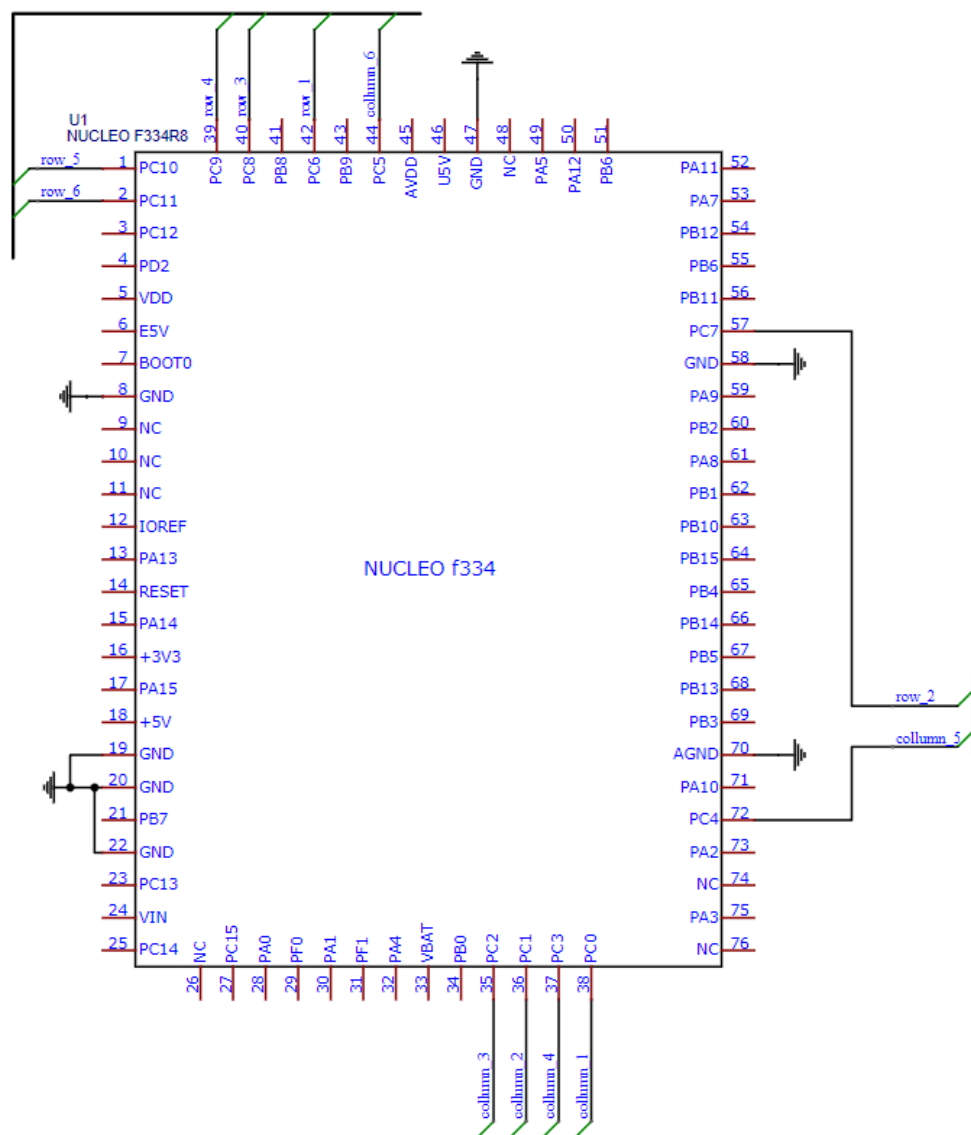


Рисунок 9. Схема подключения строк и колонок датчиков к микроконтроллеру

2.4 Заключение по разработке аппаратной части

К концу первого этапа разработки трекера был изготовлен тестовый стенд на макетной плате размером 50 мм на 100 мм (Рисунок 11). При монтаже выяснилось, что датчики достаточно хрупкие и требуют повышенной аккуратности в работе. Изготовленный стенд был успешно протестирован на наличие разрывов цепей и работоспособность, дефектов не было выявлено.

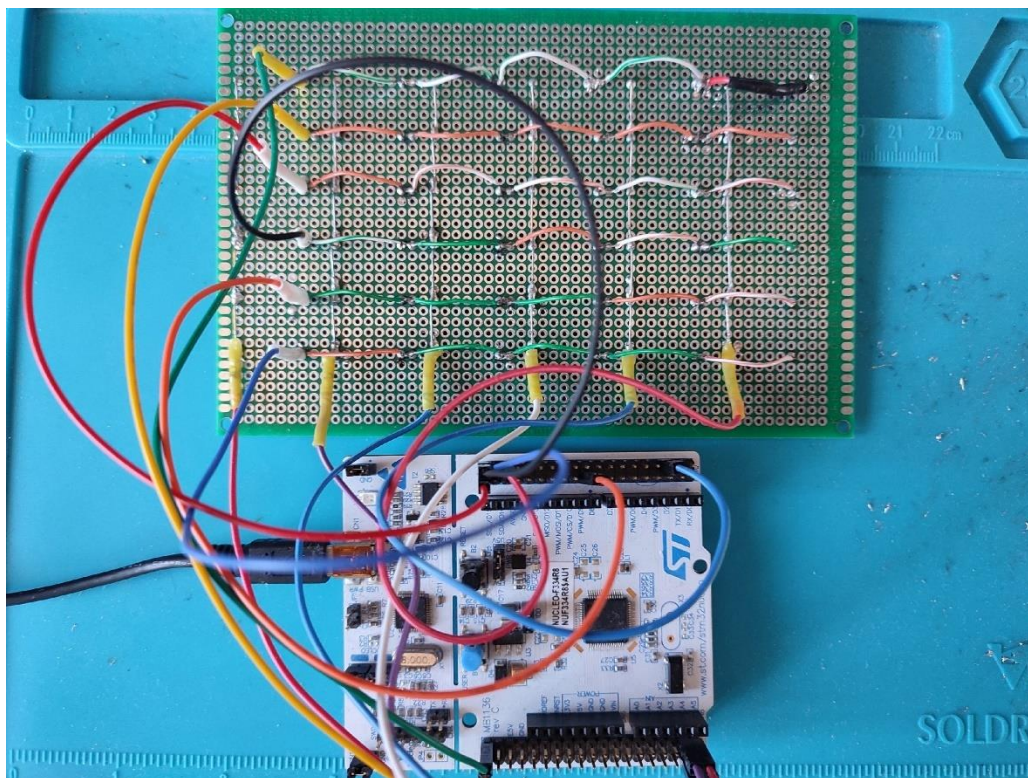


Рисунок 10. Обратная сторона спаянной платы и микроконтроллер

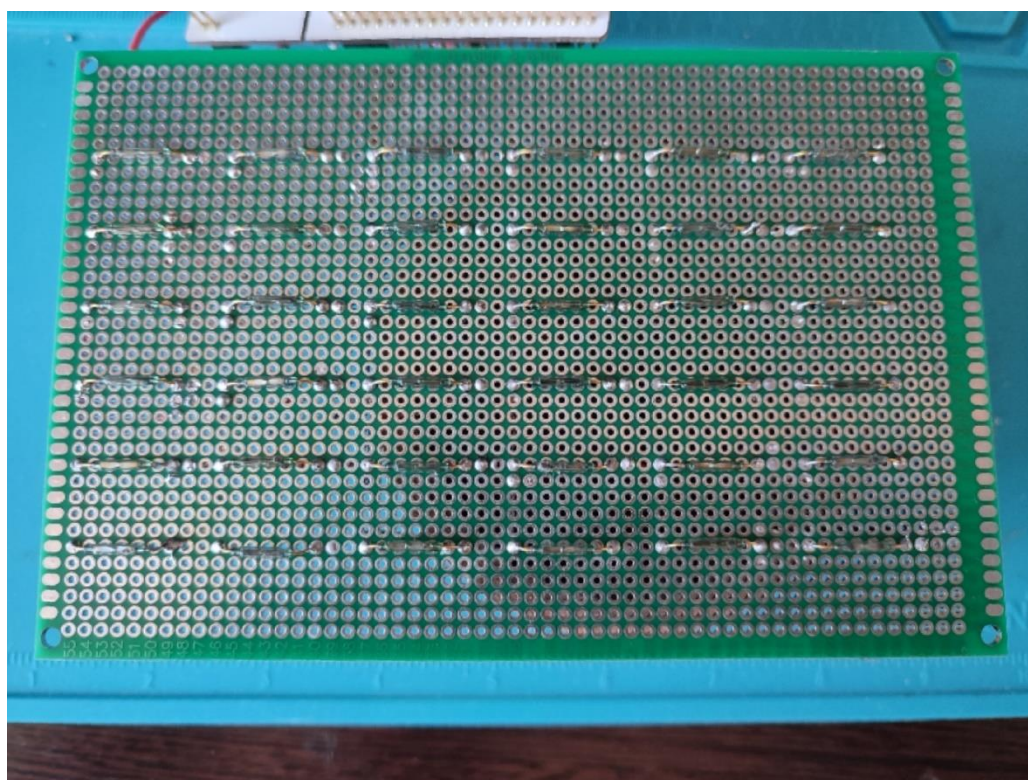


Рисунок 11. Смонтированная матрица герконов

3 Разработка программного обеспечения для трекера

3.1 Программное обеспечение для микроконтроллера STM32

3.1.1 Конфигурация устройств

Программное обеспечение проекта также состоит из двух частей. Первая часть – это программа, написанная для микроконтроллера STM32 на языке C++14 с использованием инструментария среды STM32CubeIDE версии 1.13.2. Как и в любом проекте в данной среде, сначала была задана конфигурация устройств ввода-вывода в графическом редакторе.

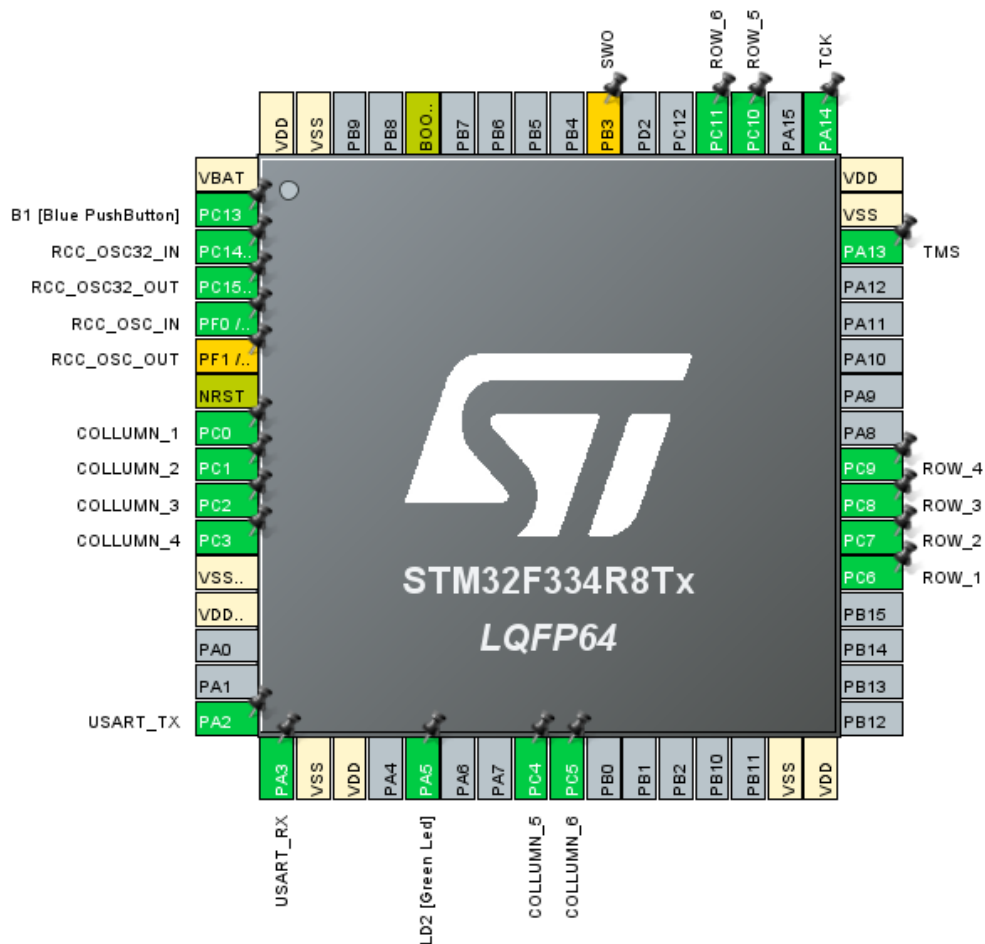


Рисунок 12. Конфигурация внешних устройств

Разделим все устройства на три группы:

- 1) UART;
- 2) цифровые выводы выбора колонки матрицы герконов;

3) цифровые входы со строк матрицы герконов.

Для первой группы устройств будет использоваться стандартный USB вывод платы микроконтроллера. На принципиальной электрической схеме платы микроконтроллера (Рисунок 13) видно, что к выводу PA2 подключён TX, а к выводу PA3 подключён RX.

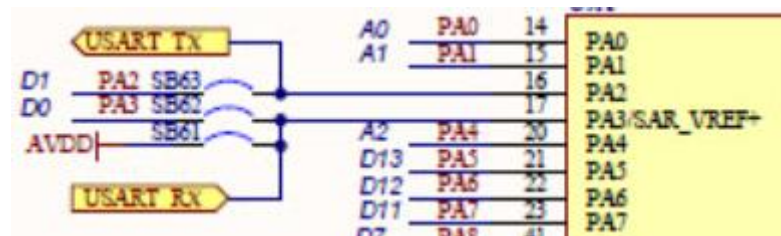


Рисунок 13. Принципиальная схема подключения UART к микроконтроллеру

Следовательно, необходимо сконфигурировать эти выводы как USART. В данном случае синхронизирующие выводы не используются, поэтому необходимо настроить UART устройство в асинхронный режим и отключить любой Flow Control. Если этого не сделать, добавятся дополнительные пины CTS/RTS, отвечающие за синхронизацию. В базовых параметрах необходимо установить baud rate равным 38400 бит/с, так как в данном случае значение частоты не влияет на работу uart. Важно, чтобы baud rate совпадал с частотой, указанной в настройках ПО для персонального компьютера.

На этапе генерации программы и конфигурации внешних устройств этой программой также генерируются структуры для работы библиотеки HAL (Hardware Abstraction Layer).

```
void SystemClock_Config(void);  
static void MX_GPIO_Init(void);  
static void MX_USART2_UART_Init(void);
```

Эти функции, сами генерируются средой разработки STM32CubeIDE. Теперь, используя сгенерированный код, необходимо реализовать функциональность разрабатываемого устройства.

3.1.2 Разработка базовой логики программы

Разработаем UML диаграмму будущих классов (Рисунок 16). Реализация классов для работы с GPIO пинами, InputPin и Output Pin, осуществлена от интерфейса Pin. В программе необходимо хранить пользовательскую конфигурацию устройства – фактическую схему того, какие именно герконы объединены в строку, какие в столбцы, а какие являются отдельными отслеживаемыми игровыми характеристиками и параметрами. Каждому изменяемому в конфигурации игровому параметру соответствует экземпляр какого-либо класса, реализующего абстрактный класс ConfigElement. Класс Matrix управляет порядком «включения» колонок и чтения строк. Некоторые классы должны работать как конечные автоматы, с постоянным вызовом некоторой функции. Такие классы будут реализовывать интерфейс IWorker.

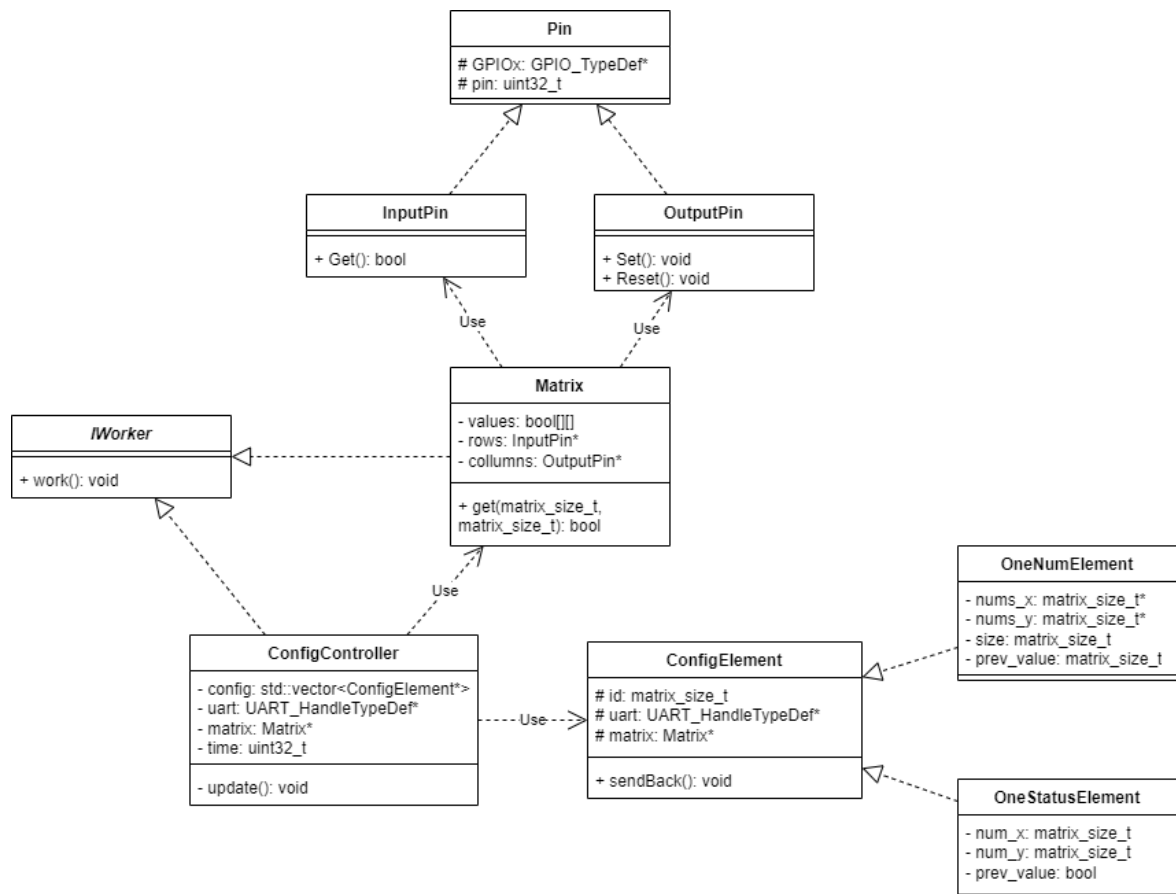


Рисунок 16. UML диаграмма классов в ПО для STM32

Реализуем интерфейс IWorker для реализации всех классов, которым необходимо вызывать свою функцию обработчик в цикле в main. Интерфейс имеет одну виртуальную функцию work(), которую будут переопределять все дочерние классы.

```

class IWorker {
public:
    virtual void work() = 0;
};
  
```

Классы управления GPIO:

```

class Pin {
protected:
    GPIO_TypeDef*    GPIOx;
    uint32_t         pin;
public:
    Pin(GPIO_TypeDef* GPIOx, uint32_t pin);
};
  
```

```

Pin::Pin(GPIO_TypeDef* GPIOx, uint32_t pin)
    : GPIOx(GPIOx), pin(pin)
{ }

```

Абстрактный класс Pin хранит параметры вывода, но не управляет выводом и вводом. Конструктор устанавливает значения игровых параметров.

Рассмотрим код одной из реализаций класса Pin – OutputPin, который управляет цифровыми выводами. В нем реализованы функции Set() и Reset(), которые используют функции HAL_GPIO_WritePin. Первая функция ставит высокий уровень на выводе, а вторая, соответственно, устанавливает низкий уровень на выводе. Конструктор класса передает игровые параметры конструктору класса родителя.

```

class OutputPin: public Pin{
public:
    OutputPin(GPIO_TypeDef* GPIOx, uint32_t pin)
        : Pin(GPIOx, pin) {}

    void Set();
    void Reset();
};

void OutputPin::Set()
{
    HAL_GPIO_WritePin(GPIOx, pin, GPIO_PIN_SET);
}

void OutputPin::Reset()
{
    HAL_GPIO_WritePin(GPIOx, pin, GPIO_PIN_RESET);
}

```

Рассмотрим класс, читающий состояние цифрового входа. В классе InputPin есть только одна функция, которая читает состояние входа через функцию HAL_GPIO_ReadPin. Конструктор устроен так же, как и в OutputPin.

```

class InputPin: public Pin{

```

```

public:
    InputPin(GPIO_TypeDef* GPIOx, uint32_t pin)
        : Pin(GPIOx, pin) {}

    bool Get();
};

bool InputPin::Get()
{
    return HAL_GPIO_ReadPin(GPIOx, pin);
}

```

Используя написанные классы для работы с портами ввода-вывода, можно реализовать класс `Matrix`, который будет обрабатывать полученные значения с физической (с датчиков) матрицы из герконов.

```

class Matrix: public IWorker {
private:
    bool values[MATRIX_SIZE][MATRIX_SIZE];
    InputPin* rows;
    OutputPin* collumns;
public:
    Matrix(InputPin rows[], OutputPin collumns[]);

    void work() override;

    bool get(matrix_size_t x, matrix_size_t y)
    { return values[x][y]; }
};

Matrix::Matrix(InputPin rows[], OutputPin collumns[])
    : rows(rows), collumns(collumns)
{
    for (size_t i = 0; i < MATRIX_SIZE; i++)
        collumns[i].Reset();
}

void Matrix::work()
{
    for (size_t i = 0; i < MATRIX_SIZE; i++)

```

```

        {
            collumns[i].Set();
            HAL_Delay(1);
            for (size_t j = 0; j < MATRIX_SIZE; j++)
                values[i][j] = rows[j].Get();
            collumns[i].Reset();
            HAL_Delay(1);
        }
    }
}

```

Логика работы данного класса: при включении одной колонки на плате в соответствии со схемой высокий сигнал будет только на тех выводах, на которых замкнут геркон. Поочерёдно считываются все строки, и считанные значения записываются в массив значений. Логика работы реализована в функции `work()`, в которой поочерёдно подается высокий сигнал на все колонки и читается каждая строка. В конструкторе инициализируются игровые параметры и все колонки выставляются на низкий уровень. Функция `get()` получает элементы формируемой матрицы.

3.1.3 Разработка протокола для взаимодействия программ

Перед разработкой классов для работы с пользовательскими конфигурациями необходимо разработать протокол, посредством которого через интерфейс USB будут коммуницировать программа на ПК и программа на микроконтроллере.

Определим, какие данные будут отправляться и приниматься. Пакет данных, отправляемый от STM32 к ПК, выглядит следующим образом:

8 bit	8 bit
ID	DATA

В поле ID содержится некоторый идентификатор параметра, отслеживаемого устройством. DATA – новое значение.

В обратную сторону будут идти данные о конфигурации устройства. Пакет будет отличаться в зависимости от того, какой конкретно игровой параметр передается. Введём два вида игровых параметров:

- 1) параметр «Состояние», который или есть, или нет;
- 2) параметр «Счётчик», который показывает значение, которое может быть цифрой для пользователя;

Для каждого параметра состояния на устройство будет передан следующий пакет:

8 bit	8 bit	8 bit	8 bit
1	ID	X	Y

Первый байт – указание на тип пакета, ID – ID параметра. X – номер колонки в матрице, Y – номер строки в матрице, оба соответствующие геркону, состояние которого считать состоянием игрового параметра.

Для параметра счётчика на устройство будет передан следующий пакет:

8 bit	8 bit	8 bit	8 bit	8 bit	8 bit	8 bit	...
0	ID	Size	X1	Y1	X2	Y2	...

Первый байт также является указателем на тип пакета. Аналогично с предыдущим вариантом осталось и для ID. Бит size показывает, сколько возможных чисел может быть в данном параметре, соответственно, и сколько герконов задействовано в его определении. Далее идут size пар чисел X_i и Y_i , которые показывают, какие строки и столбцы матрицы соответствуют числу i .

После приема всех частей конфигураций приходит число 0xFF, показывающее, что вся конфигурация передана.

Микроконтроллеру не обязательно знать имя игрового параметра, так как для идентификации конкретного параметра существует ID, поэтому нет нужды передавать на устройство сложные типы данных, такие как строки.

В основе пользовательской конфигурации лежит абстрактный класс ConfigElement.

```
class ConfigElement {
protected:
    matrix_size_t id;
    UART_HandleTypeDef* uart;
    Matrix* matrix;

public:
    ConfigElement(matrix_size_t id, UART_HandleTypeDef* uart, Matrix*
matrix);

    virtual void sendBack() = 0;
};

ConfigElement::ConfigElement(matrix_size_t id, UART_HandleTypeDef* uart,
Matrix* matrix)
: id(id), uart(uart), matrix(matrix)
{ }
```

В конструкторе устанавливаются поля и создается виртуальная функция sandBack(), которая отправляет данные считанного игрового параметра на ПК. Параметр ID должен быть у каждого элемента конфигурации, по нему сопоставляется полученное значение и конкретный параметр в программе на персональном компьютере.

```
class OneNumElement: public ConfigElement {
private:
    matrix_size_t* nums_x;
    matrix_size_t* nums_y;
    matrix_size_t size;

    matrix_size_t prev_value;
public:
    OneNumElement(matrix_size_t id, UART_HandleTypeDef* uart, Matrix*
matrix, matrix_size_t* nums_x, matrix_size_t* nums_y, matrix_size_t size);
    ~OneNumElement();
    void sendBack() override;
```

```

};

OneNumElement::OneNumElement(matrix_size_t id, UART_HandleTypeDef* uart,
Matrix* matrix, matrix_size_t* nums_x, matrix_size_t* nums_y, matrix_size_t size)
: ConfigElement(id, uart, matrix), nums_x(nums_x), nums_y(nums_y),
size(size), prev_value(MATRIX_SIZE * MATRIX_SIZE + 1)
{ }

void OneNumElement::sendBack()
{
    matrix_size_t data = 0;
    for (ssize_t i = size - 1; i >= 0; i--)
    {
        if (matrix->get(nums_x[i], nums_y[i]))
        {
            data = i + 1;
        }
    }
    if (data != prev_value)
    {
        HAL_UART_Transmit(uart, &id, sizeof(matrix_size_t), 500);
        HAL_UART_Transmit(uart, &data, sizeof(matrix_size_t), 500);
        prev_value = data;
    }
}

OneNumElement::~OneNumElement()
{
    free(nums_x);
    free(nums_y);
}

```

Рассмотрим класс oneNumElement, который обрабатывает параметр «Счётчики». В функции sendBack() проверяются все герконы, которые относятся к данному счётчику, и, если с прошлой итерации число изменилось, то новое число передаётся на ПК.

Другой класс, OneStatusElement, работает примерно так же, но читает данные только из одного элемента матрицы. В функции sendBack()

считывается значение из матрицы, и если оно изменилось, то новое значение посылается на ПК.

Теперь разберем класс, который управляет всей пользовательской конфигурацией:

```
class ConfigController: public IWorker
{
private:
    std::vector<ConfigElement*> config;
    UART_HandleTypeDef* uart;
    Matrix* matrix;
    uint32_t time;
public:
    ConfigController(UART_HandleTypeDef* uart, Matrix* matrix);

    void work() override;
private:
    void update();
};

void ConfigController::work()
{
    if (__HAL_UART_GET_FLAG(uart, UART_FLAG_RXNE))
        update();

    if (HAL_GetTick() - time > TIMEOUT)
    {
        time = HAL_GetTick();
        for (auto &element: config)
            element->sendBack();
    }
}

void ConfigController::update()
{
    config.clear();
    while(1)
    {
        matrix_size_t data;
```

```

        HAL_UART_Receive(uart, &data, sizeof(matrix_size_t), 500);
        switch (data)
        {
        case END_ID:
            return;
        case ONE_NUM_ELEMENT_ID:
        {
            matrix_size_t id;
            HAL_UART_Receive(uart, &id, sizeof(matrix_size_t), 500);

            matrix_size_t size;
            HAL_UART_Receive(uart, &size, sizeof(matrix_size_t), 500);
            matrix_size_t* nums_x = (matrix_size_t*)malloc(size *
sizeof(matrix_size_t));
            matrix_size_t* nums_y = (matrix_size_t*)malloc(size *
sizeof(matrix_size_t));

            for (matrix_size_t i = 0; i < size; i++)
            {
                HAL_UART_Receive(uart,      nums_x      +      i,
sizeof(matrix_size_t), 500);
                HAL_UART_Receive(uart,      nums_y      +      i,
sizeof(matrix_size_t), 500);
            }

            OneNumElement* new_element = new OneNumElement(id, uart,
matrix, nums_x, nums_y, size);

            config.push_back(dynamic_cast<ConfigElement*>(new_element));
            break;
        }
        case ONE_STATUS_ELEMENT_ID:
        {
            matrix_size_t id;
            HAL_UART_Receive(uart, &id, sizeof(matrix_size_t), 500);

            matrix_size_t num_x;
            HAL_UART_Receive(uart,  &num_x,  sizeof(matrix_size_t),
500);

            matrix_size_t num_y;

```

```

        HAL_UART_Receive(uart, &num_y, sizeof(matrix_size_t),
500);

        OneStatusElement* new_element = new OneStatusElement(id,
uart, matrix, num_x, num_y);

        config.push_back(dynamic_cast<ConfigElement*>(new_element));
            break;
        }
        default:
            return;
        }
    }

}

ConfigController::ConfigController(UART_HandleTypeDef* uart, Matrix* matrix)
: uart(uart), matrix(matrix), time(0)
{ }

```

Класс имеет приватный метод update, который получает от ПК пользовательскую конфигурацию и обновляет конфигурацию config. Метод work() проверяет, что в uart есть данные, и, если они действительно есть, вызывает функцию update для обновления конфигурации. Если прошло необходимое время, то по очереди вызывается каждый элемент конфигурации. Функция update сначала очищает старая конфигурация. Далее в цикле читается один байт до тех пор, пока не получено число 0xFF, обозначающее конец пакета. После побайтового получения параметров конфигурации создаётся объект, добавляемый в вектор, в котором хранится пользовательская конфигурация.

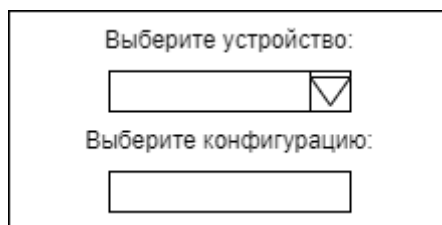
За счет удобной структуры ПО для микроконтроллера для добавления нового типа параметра достаточно просто реализовать соответствующую реализацию класса Config и добавить способ чтения данного элемента через UART.

Для запуска создаются объекты всех классов, передаются нужные игровые параметры, а в бесконечном цикле вызываются все необходимые `work()`. Программа написана в объектно-ориентированном стиле, с разбиением кода на заголовочные файлы `.h` и исполняемые `.cpp`.

3.2. Графический интерфейс

3.2.1 Создание страниц UI

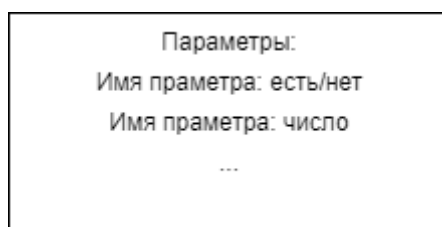
Для второй части, работающей на стороне ПК ведущего игрока, необходимо сделать простой графический интерфейс, состоящий из двух страниц. Первая будет с выбором текущей пользовательской конфигурации и выбором конкретного устройства (Рисунок 17).



The image shows a graphical user interface for the first page. It contains two labels: "Выберите устройство:" (Select device:) and "Выберите конфигурацию:" (Select configuration:). Below the first label is a text input field followed by a dropdown arrow icon. Below the second label is a text input field.

Рисунок 17. UI первой страницы

Вторая страница в свою очередь будет изменяться в зависимости от выбранной конфигурации (Рисунок 18). Чем больше игровых параметров отслеживается, тем больше должна быть страница.



The image shows a graphical user interface for the second page. It contains a label "Параметры:" (Parameters:). Below it are two lines of text: "Имя параметра: есть/нет" (Parameter name: yes/no) and "Имя параметра: число" (Parameter name: number). Below these is a horizontal ellipsis "..." indicating more parameters.

Рисунок 18. UI второй страницы

После выбора конфигурации, данный выбор должен отправляться на устройство. Порт, к которому подключено устройство, прослушивается и при обновлении данных на стороне трекера обновляет их и на странице. Для реализации подобной задачи выбран язык Python. Пользовательский интерфейс реализован с использованием библиотеки `tkinter`.

С помощью функции `serial.tools.list_ports.components()` из библиотеки `serial` получается список всех доступных портов, но в виде компонент. Для получения списка строк необходимо через функцию `map()` получить `device` каждого из портов:

```
ports = serial.tools.list_ports.comports()
ports_str = list(map(lambda x: x.device, ports))
```

Для отображения надписи с информацией о том, что пользователю нужно выбрать порт, необходимо использовать объект `label`. Отрисовка происходит за счёт следующих строк кода:

```
label = Label(text="Выберите устройство:")
label.pack()
```

По полученным существующим портам создаётся блок с выбором вариантов. В данной библиотеке он называется `Combobox`. Создание и отрисовка выпадающего списка со всеми портами, которые подключены к ПК:

```
box = ttk.Combobox(values=ports_str)
box.pack(anchor=NW, padx=6, pady=6)
```

Для создания текстового поля для подгрузки пользовательской конфигурации в библиотеке существует класс `Text`. Необходимо указать высоту поля, так как по умолчанию поле высотой четыре строки. Часть программы, отвечающая за это, выглядит следующим образом:

```
filename = Text(height=1, wrap="char")
filename.pack()
```

Для работы кнопки перехода на следующую страницу UI необходимо написать функцию-обработчик, которая будет вызываться после нажатия. В этом обработчике должно сохраняться введённое значение выпадающего списка и текстового поля, должны уничтожаться описанные выше компоненты и вызываться функция отрисовки второго окна. Используя функции `get()` и `destroy()`, напомним функцию-обработчик, далее создадим компонент кнопки и отрисуем её:

```

def button():
    port = box.get()
    fl = filename.get("1.0", "end")
    print(port, fl)
    label.destroy()
    box.destroy()
    label2.destroy()
    filename.destroy()
    button.destroy()
    connect_port(fl, port)

button = Button(text="Старт", command=button)
button.pack()

root.mainloop()

```

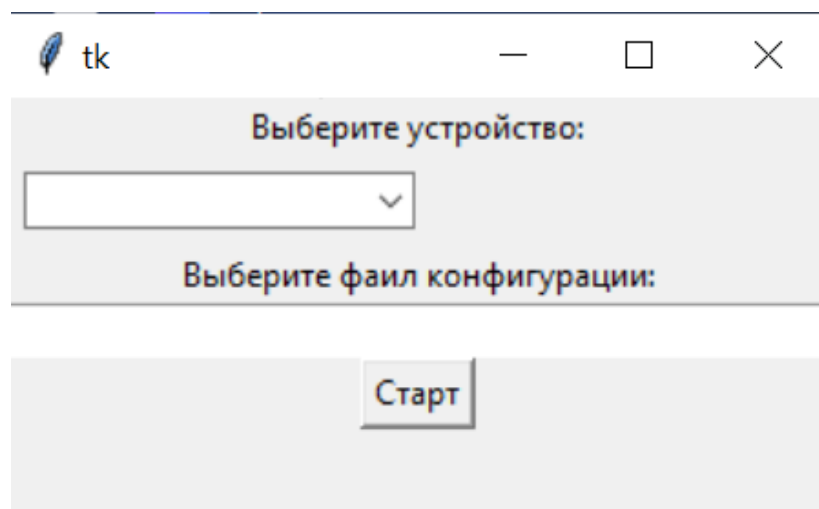


Рисунок 19. Изображение полученной страницы

3.2.2 Работа с пользовательской конфигурацией

Перед тем, как продолжать работать с кодом второй страницы, разберемся с хранением пользовательской конфигурации на компьютере. Файл конфигурации представлен текстовым файлом следующего формата:

Тип	ID	Имя	Данные об элементе
-----	----	-----	--------------------

Данные зависят от того, какого типа полученный элемент. Для элемента типа «Счётчик» данные выглядят следующим образом:

Size	X1	X2	...	Y1	Y2	...
------	----	----	-----	----	----	-----

Такое расположение связано с удобством чтения в языке Python. Для элемента типа «Состояние» конфигурация будет проще:

X	Y
---	---

Таким образом, полная строка пользовательской конфигурации типа «Счётчик» выглядит следующим образом:

```
0 0 hp 3 0 0 0 1 2 3
```

Данный пример конфигурации соответствует счётчику от 1 до 3, в котором цифра 1 – это геркон с координатам {0;1}, 2 – геркон с координатами {0;2}, а 3, соответственно, с координатами {0;3}.

Конфигурация для «Состояния» выглядит следующим образом:

```
1 1 fire 1 0
```

Данный пример соответствует геркону с координатой {1;0}.

Обратимся к функции чтения пользовательской конфигурации из файла:

```
def read_config(config):
    global ONE_NUM_ELEMENT_ID, ONE_STATUS_ELEMENT_ID, END_ID

    config_file = open(config, "r")
    conf = []
    while line := config_file.readline():
        splitted_line = line.split()
        if splitted_line[0] == str(ONE_NUM_ELEMENT_ID):
            conf.append({"type": splitted_line[0],
                        "id": splitted_line[1],
                        "name": splitted_line[2],
                        "size": splitted_line[3],
                        "num_x": [splitted_line[i] for i in range(4, 4 +
int(splitted_line[3]))],
                        "num_y": [splitted_line[i] for i in range(4 +
int(splitted_line[3]), 4 + int(splitted_line[3]) + int(splitted_line[3]))])
        else:
            conf.append({"type": splitted_line[0],
                        "id": splitted_line[1],
                        "name": splitted_line[2],
                        "num_x": splitted_line[3],
                        "num_y": splitted_line[4]})
    return conf
```

На выход функции приходит имя файла с пользовательской конфигурацией. В функции поочерёдно читаются все строки, каждая делится пробелами, а затем из разделённой строки извлекаются необходимые данные. Для удобства работы num_x и num_y извлекаются по следующему алгоритму: splitted_line[i] for i in range(4, 4 + int(splitted_line[3])) и

splitted_line[i] for i in range(4 + int(splitted_line[3]), 4 + int(splitted_line[3]) + int(splitted_line[3])) соответственно. Данные хранятся не так, как в протоколе, для удобства получения координаты.

Рассмотрим функцию для загрузки пользовательской конфигурации на STM32.

```
def write_config(conf, port):
    global ONE_NUM_ELEMENT_ID, ONE_STATUS_ELEMENT_ID, END_ID

    for line in conf:
        port.write(bytes([int(line['type'])]))
        port.write(bytes([int(line['id'])]))
        if int(line['type']) == ONE_NUM_ELEMENT_ID:
            port.write(bytes([int(line['size'])]))
            for i in range(int(line['size'])):
                port.write(bytes([int(line['num_x'])[i])])
                port.write(bytes([int(line['num_y'])[i])])

        if int(line['type']) == ONE_STATUS_ELEMENT_ID:
            port.write(bytes([int(line['num_x'])]))
            port.write(bytes([int(line['num_y'])]))

    port.write(bytes([END_ID]))
```

На вход функции приходит конфигурация, полученная при чтении файла, и открытый СОМ порт из библиотеки pyserial. В функции поочередно обрабатывается каждый элемент полученной конфигурации в зависимости от типа элемента. На микроконтроллер отправляются данные конфигурации в соответствии с разработанным ранее протоколом общения. Используя разработанные ранее функции, инициализируется новая страница:

```
conf = read_config(config.strip())

ser = serial.Serial(port, baudrate=38400)
write_config(conf, ser)
```

Сначала читается конфигурация, затем открывается serial порт с baud rate, совпадающим с тем, что был выбран в конфигурации STM32. Далее считанная конфигурация отправляется на устройство и создаётся новая страница. Отрисовать полученную информацию в UI не получится до тех пор, пока не будет получена информация с STM32. Для получения данных написана специальная функция, которая асинхронно будет вызываться раз в

некоторый промежуток времени. В этой функции будет читаться COM порт и меняться значение соответствующей данному параметру подписи:

```
if ser.inWaiting():
    id = ser.read(1)
    data = ser.read(1)
    print(id, data)
    for lab in range(len(lables)):
        if bytes([int(conf[lab]['id'])]) == id:
            if int(conf[lab]['type']) == ONE_STATUS_ELEMENT_ID:
                lables[lab].config(text=f"{conf[lab]['name']}: "
                                     f"'да' if int.from_bytes(data,
byteorder='little') else 'нет'}")
            else:
                lables[lab].config(text=f"{conf[lab]['name']}: "
                                     f"{int.from_bytes(data,
byteorder='little')}}")
```

Сначала читается ID элемента, который был изменён, затем читается новое значение этого элемента. Далее осуществлён проход по всей полученной конфигурации и, если id найден, проверяется тип записи. В зависимости от типа будет установлен разный текст в соответствующую запись.

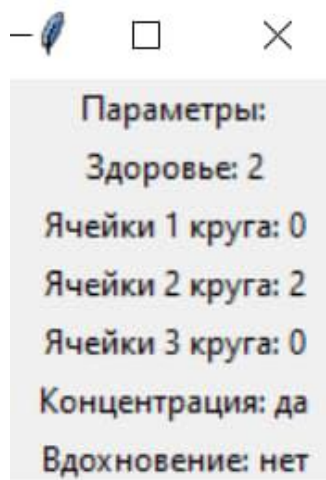


Рисунок 20. Страница с собираемыми параметрами

Таким образом, реализованная вторая страница позволяет отображать в отдельном окне цифровые и бинарные параметры. Как видно, счетчики пишут рядом с названием параметра число, а состояния пишут «да» или «нет», в зависимости от состояния соответствующего геркона (замкнут или разомкнут).

3.3. Заключение по разработке программного обеспечения

Для разрабатываемого для ВКР устройства было разработано ПО, состоящее из двух частей, связанных через последовательный интерфейс. Первая часть – программа, реализованная на языке C++ для платформы STM32. Для упрощения процесса разработки был применен графический редактор конфигурации выводов в среде STM32CubeIDE, для работы с периферией использовалась библиотека HAL. Это ПО использовалось для сбора информации с матрицы герконов, получения конфигурации устройства и отслеживания изменений состояний параметров конфигурации. Когда ПО обнаруживает, что в параметре произошло изменение, его новое значение отправляется на ПК с указанием ID элемента.

Вторая часть программы представляет из себя программу на Python с использованием библиотеки tkinter для реализации графического интерфейса. Программа имеет две страницы. Первая – выбор подключенных устройств и выбор файла конфигурации. Вторая отображает считанные игровые параметры.

Для взаимодействия двух программ был разработан протокол передачи данных, для хранения пользовательских конфигураций на ПК был разработан формат хранения файлов конфигурации в текстовых файлах.

4 Тестирование и определение характеристик

В ходе работы над ВКР была разработана принципиальная электрическая схема платы и ПО. На основе разработанной схемы была собрана тестовая плата с матрицей из 36 герконов (Рисунок 11).

К разработанной плате была подключена тестовая плата STM32 NucleoF334. В результате при подключении USB ко входу тестовой платы можно считать, какие из герконов были замкнуты, а какие нет. При поднесении магнита к датчику контакт замыкается, что отображается в графическом интерфейсе на ПК ведущего игрока. Таким образом, в результате функционального тестирования изготовленного трекера получен работающий прототип разрабатываемого устройства.

К сожалению, в ходе разработки была выявлена проблема данного способа подключения матриц герконов, а именно фантомные нажатия. Пример возникновения фантомного нажатия можно увидеть на рисунке (Рисунок 21).

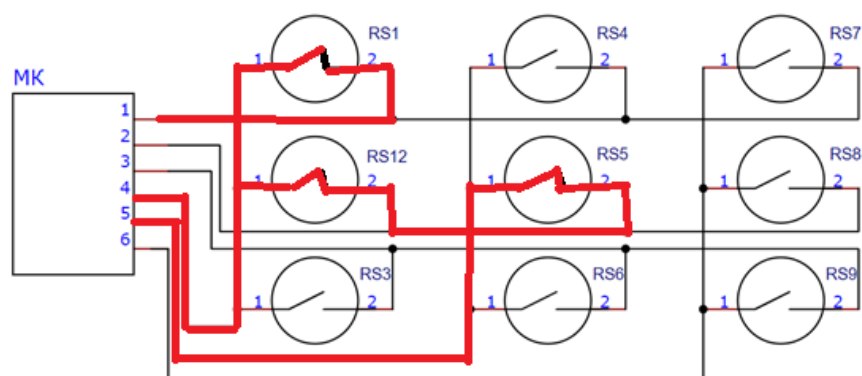


Рисунок 21. Возникновение фантомных нажатий

В данном примере матрица сканирует первую строку. Предположим, что замкнут один геркон в первой строке и два во второй, как показано на рисунке. Тогда ток пройдет через замкнутый геркон RS1 в RS12 и через него попадёт в RS5. Тогда система определит, что замкнуты сразу два геркона, RS1 и RS4, хотя в первой строке замкнут лишь один. Подобная проблема усложняет

использование устройства, но так как большинство параметров представляют из себя счётчик, в котором не может быть больше одного замыкания в строке, в большинстве случаев подобная проблема и не возникнет. Необходимо учитывать эту проблему при разработке пользовательских конфигураций и определять считывание параметров так, чтобы не допускать фантомного срабатывания. Иной способ решения проблемы – написание редактора конфигураций, запрещающего использовать такие сочетания. Или можно решить данную проблему путём изменения способа подключения матрицы на любой из описанных в рамках данной ВКР, что потребует серьёзных трудо- и ресурсозатрат. Тем не менее, помимо одного реализованного в проекте способа подключения герконов так же был реализован и метод подключения, основанный на сумме токов, в итоговой же презентации представлен способ подключения по строкам и столбцам как более надёжный и проверенный.

4.1 Определение физических характеристик трекера

Размер итогового устройства составляет 100 мм на 150 мм, питание устройства происходит за счёт питания от ПК ведущего игрока, получаемого по USB интерфейсу. Замеренное токопотребление в среднем составляет 0.14А. Это выше, чем в среднем у USB устройств, но связано это с тем, что основными потребителями энергии в подобных устройствах, если они не имеют движущихся частей, являются микроконтроллеры. Так как используемая в данном проекте плата NucleoF334 имеет два микроконтроллера, основной и программатор, то и происходит повышение токопотребления.

4.2 Определение пользовательских характеристик

В ходе тестирования и работы с устройством необходимо было понять, насколько удобно использовать разработанный прототип трекера.

Время отклика невелико и визуально не заметно. Это связано с тем, что устройство передаёт данные не с определенной частотой, а как только

происходят изменения в значениях. Поэтому частота обновления данных связана только с частотой работы основного цикла в ПО STM32. С точки зрения пользователя, опираясь на требования предметной области, время отклика не должно быть превышать 0.5 мс, так как в среднем за этот промежуток игрок может перенести магнитную метку на другой контакт, тем самым обманывая других игроков путём нарушения правил игры.

Разработанный прототип удобно подключать и питать, так как требуется только один работающий ПК, являющийся и источником питания для всех существующих в системе трекеров.

Из того, что может оказаться для пользователя не очень удобными и понятными, – это составление своих пользовательских конфигураций. Так как игровых систем достаточно много, а трекер заявлен универсальным, конфигурации необходимо составлять вручную в не самом удобном для восприятия человека формате. Эту проблему возможно решить написанием еще одной дополнительной программы редактора пользовательских конфигураций, но в рамках ВКР она не была реализована.

Цена на разработку по сравнению с аналогами также существенно ниже:

- микроконтроллер (в среднем) – 3500 рублей;
- 36 датчиков герконов – 1296 рублей;
- макетная плата – 400 рублей;
- провода – 20 рублей.

Суммарно: 5216 рублей. Стоимость можно снизить за счёт производства печатной платы на заводе, но изготовление увеличится по срокам. Аналоговый трекер выходит ниже по цене, но так как он не обладает универсальностью, то проигрывает по той причине, что для достижения такой же универсальности придётся потратить даже больше средств на покупку нескольких трекеров.

Таким образом, если уже существует подходящая под задачи конфигурация, то использование устройства будет комфортным и быстрым, задержка незначительна и не создает дискомфорта игрокам.

ЗАКЛЮЧЕНИЕ

В ходе выполнения ВКР был проанализирован рынок автоматизированных аксессуаров для настольных игр и выявлены плюсы и минусы аналогов существующих трекеров игровых параметров, также сформулированы требования для разрабатываемого устройства.

Опираясь на сформулированные требования к устройству и ПО, была разработана плата устройства. Для этого были проанализированы способы подключения матриц герконов к микроконтроллеру, из них был выбран метод подключения датчиков по строкам и столбцам. Для реализации прототипа также был выбран микроконтроллер STM32 NucleoF334, так как его параметры подходят по установленным критериям.

Для разработанной платы было реализовано ПО, состоящее из двух частей – ПО для контроллера STM32 NucleoF334 и для ПК. Первая программа собирает данные с матрицы герконов и передает их по разработанному протоколу на ПК. Программа реализована в объектно-ориентированном стиле и может легко быть расширена, если будут необходимы новые элементы пользовательской конфигурации. Вторая программа представляет из себя графический интерфейс, который собирает данные с устройства и отображает их ведущему игроку. Интерфейс имеет две страницы, на первой выбирается конкретное устройство и конфигурация, на второй отображаются игровые параметры из пользовательской конфигурации. Также был разработан формат хранения конфигураций, позволяющий реализовывать ту конфигурацию, которая необходима под конкретную пользовательскую задачу.

Разработанное устройство было испытано, определены объективные физические, и субъективные пользовательские характеристики. Из тестов можно сделать вывод, что устройство имеет приемлемые для работы физические характеристики, а пользовательский опыт в целом положительный, но имеется существенный минус – сложность создания

собственных пользовательских конфигураций и вероятность возникновения фантомных нажатий.

Таким образом, в ходе выполнения ВКР были выполнены все поставленные цели и, как итог, разработан прототип устройства, которое может применяться для решения поставленной в рамках ВКР цели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Luo X. Design of Portable ECG Device Based on STM32 //2016 6th International Conference on Machinery, Materials, Environment, Biotechnology and Computer. – Atlantis Press, 2016. – С. 386-389.
2. Tan Z. et al. Design of Smart Card Chip Reader Based on STM32 //Journal of Networking and Telecommunications. – 2020. – Т. 2. – №. 1. – С. 7–11.
3. А.О. Ключев, Д.Р. Ковязина, П.В. Кустарев, А.Е. Платунов. Аппаратные и программные средства встраиваемых систем – Санкт-Петербург: Университет ИТМО, 2010. – 290 с.
4. А.О. Ключев, Д.Р. Ковязина, Е.В. Петров, А.Е. Платунов. Интерфейсы периферийных устройств. – Санкт-Петербург: СПбГУ ИТМО, 2010. – 294 с.
5. Документация STM32F334xx [Электронный ресурс] – URL: <https://www.st.com/resource/en/datasheet/stm32f334k4.pdf> (дата обращения: 20.05.2024)
6. Составное устройство USB на STM32 [Электронный ресурс] – URL: <https://habr.com/ru/articles/532038/> (дата обращения: 20.05.2024)
7. USART vs UART: Know the difference [Электронный ресурс] – URL: <https://www.radiolocman.com/review/article.html?di=164400> (дата обращения: 20.05.2024)
8. Руководство по Tkinter [Электронный ресурс] – URL: <https://metanit.com/python/tkinter/> (дата обращения: 20.05.2024)
9. ГОСТ 2.702-2011 «Правила оформления электрических схем»
10. Role-play Games (RPGs) for Mental Health (Why Not?): Roll for Initiative [Электронный ресурс] – URL:

<https://link.springer.com/article/10.1007/s11469-022-00832-y> (дата обращения:
20.05.2024)