Национальный исследовательский ядерный университет «МИФИ»

УДК № FIXME
Регистрационный № FIXME
Инв. №

УТВЕРЖДАЮ

Зав. кафедрой Электроники

_____ Барбашов В.М.

«_____» _____ 2017 г.

ОТЧЁТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

СИСТЕМА ЗАДАНИЯ ТЕМПЕРАТУРЫ ИС ПРИ ПРОВЕДЕНИИ РАДИАЦИОННЫХ ИСПЫТАНИЙ НА ВОЗДЕЙСТВИЕ ОЯЧ

(заключительный)

г. Москва, 2017

РЕФЕРАТ

Ключевые слова АВТОМАТИЗАЦИЯ, РАДИАЦИОННЫЕ ИСПЫТАНИЯ, КОНТРОЛЬ ТЕМПЕРАТУРЫ, МИКРОКОМПЬЮТЕРЫ

Отчет содержит 60 стр. 7 рис. 9 таблиц.

Целью работы является разработка автоматизированной системы и выявление наиболее оптимальных методик измерения параметров аналого-цифровых блоков сложно-функциональных СБИС при проведении радиационных испытаний.

FIXME результаты, эффективность и область применения, выводы, основные конструктивные и технико-эксплуатационные характеристики.

FIXME факультативная информация?

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Большое целое — целое число с неограниченной разрядностью.

Обратный прокси — тип прокси-сервера, который ретранслирует запросы клиентов из внешней сети на один или несколько серверов внутренней сети.

Одноплатный микрокомпьютер — маленький, относительно недорогой компьютер с микропроцессором в качестве CPU. «Одноплатный» означает, что микрокомпьютер реализован в виде одной печатной платы, без интегрированной периферии вроде клавиатуры либо экрана.

ТРИД — ПИД-регулятор температуры двухканальный ТРИД РТП322, разработанный ООО «Вектор-ПМ».

ОПРЕДЕЛЕНИЯ

В настоящем отчете о НИР применяют следующие термины с соответствующими определениями.

ОЯЧ — отдельные ядерные частицы.

ИС — интегральная схема.

ЯП — язык программирования.

СБИС — сверхбольшая интегральная схема.

ПО — программное обеспечение.

ОС — операционная система.

API — application programming interface — внешний интерфейс взаимодействия с приложением.

PyPI — Python Package Index — официальный репозиторий с библиотеками Python.

ВВЕДЕНИЕ

Целью работы является разработка элементов автоматизированной системы и выявление наиболее оптимальных методик измерения параметров аналого-цифровых блоков сложно-функциональных СБИС при проведении радиационных испытаний. В настоящий момент контроль за температурой СБИС при проведении радиационных испытаний осуществляется посредством системы из находящихся в одном корпусе ПИД-регулятора ТРИД РТП322 и блока питания для нагревательных элементов, и находящихся вне корпуса нагревательных элементов совмещённых с измерительной термопарой. Настройка системы осуществляется вручную путём использования имеющейся у ТРИД лицевой панели, либо через подключённый по RS485 персональный компьютер с программой на ЯП LabVIEW.

Оба варианта управления не отличаются удобством: для использования лицевой панели необходимо находиться в месте проведения испытания, кроме того интерфейс из четырёх физических кнопок проигрывает по своим характеристикам (время задания параметра оператором, вероятность совершения ошибки оператором при задании параметра) возможному программному интерфейсу. Для использования персонального компьютера, подключённого по RS485 необходимо, чтобы указанный компьютер был размещён в месте проведения испытания и подключён к ТРИД. Указанный персональный компьютер может управляться удалённо, с использованием TeamViewer или аналогичного ПО.

Разрабатываемая автоматизированная система предполагает размещение управляющего модуля в одном корпусе с ПИД-регулятором.

Для достижения поставленной цели необходимо решить следующие задачи:

— Определение способа удалённого доступа и подбор управляющего модуля.

— Определение технологий, используемых для написания программного обеспечения управляющего модуля.

— Создание макета системы с размещённым управляющим модулем и необходимых дополнительных элементов (системы питания управляющего модуля, проводов для подключения управляющего модуля к ТРИД).

— Создание программного обеспечения, реализующего управление ТРИД.

— Проведение испытаний с использованием макета и устранение выявленных недостатков.

— FIXME

1 Аналитический раздел

2 Конструкторский раздел

2.1 Определение способа удалённого доступа и подбор типа управляющего модуля

Комплекс, в котором проходят испытания, имеет два основных физических разделённых модуля:

— место проведения испытаний, в котором находится камера для проведения испытаний и бóльшая часть оборудования

— и операторская, в которой находятся компьютеры, предоставляющие доступ к оборудованию.

Между модулями присутствует связь в виде одноранговой локальной сети, доступен WiFi (IEEE 802.11a/b/g) и Ethernet (IEEE 802.3i/y/ab) в месте проведения испытаний. Возможность создания дополнительных способов связи посредством протягивания новых кабелей или установки собственных ретрансляторов для беспроводной связи не предусмотрена, так же отсутствует беспроводная связь с внешним миром.

FIXME ссылки на стандарты

В связи с характером радиационных испытаний нахождение человека в месте проведения испытаний является нарушением техники безопасности, поэтому необходимо осуществление удалённого управления. В указанных условиях единственным доступным способом удалённого управления является использование предоставляемой локальной сети, поэтому первым требованием к управляющему модулю является поддержка TCP/IP стёка.

Выбор между подключением по WiFi и подключением по Ethernet определяется объёмом траффика между управляющим модулем и компьютером оператора, допустимой задержка реакции системы на команды оператора и допустимой задержкой при передаче измеренных значений температуры. Рассмотрим этот вопрос подробнее:

— Объём траффика — для контроля за ходом испытаний достаточно раз в секунду снимать показания температурного сенсора с ТРИД и иногда присылать новые значения параметров. ТРИД поддерживает только 16-битовые целые, что даёт объём траффика от управляющего модуля не менее 32 бит в секунду. Установка параметров с определённым периодом не требуется, поэтому требования к сетевому подключению определяются не их объёмом, а только допустимой задержкой реакции на команды оператора.

— Допустимая задержка реакции системы на команды оператора — в связи с тем, что в ходе одного испытания не предполагается изменение параметров

системы, которое должно происходить между испытаниями, в отсутствие форс-мажорных обстоятельств вполне допустима задержка реакции на одну минуту. При необходимости срочного отключения нагрева данную операцию требуется осуществить не менее чем за секунду.

— Допустимая задержка при передаче измеренных значений температуры составляет половину секунды. Данные значения записываются в журнал испытаний и необходимы для соотнесения колебаний температуры с отказами системы.

FIXME найти или сделать исследование задержек WiFi и Ethernet сетей. Или как-нибудь поаккуратнее проигнорировать.

Как видно, при данных условиях возможно осуществление подключения как по WiFi сети, так и с использованием Ethernet. Однако подключение по WiFi сети облегчает процедуру сбора испытательного стенда, поэтому следующим требованием к управляющему модулю служит наличие возможности подключения по WiFi.

Также в связи с малыми объёмами трафика допустимо использование HTTP протокола и сериализация передаваемых данных не наиболее оптимальным, а наиболее удобным способом. Таким образом можно задействовать браузер для отображения интерфейса и взаимодействия с управляющим модулем, не тратя ресурсы на написание отдельного приложения.

ТРИД имеет единственный интерфейс подключения к управляющему модулю[1]: протокол modbus, работающий поверх UART через интерфейс RS485, что требует наличия RS485 у управляющего модуля. Также желательно наличие готовых библиотек для работы с modbus. В ходе исследования выяснилось, что ТРИД можно модифицировать для поддержки полнодуплексного UART по трём линиям: RX/TX/GND с высоким уровнем равным 5 В.

Таким образом, управляющий модуль должен соответствовать следующим требованиям:

— Помещаться в одном корпусе с ТРИД.

— Поддерживать TCP/IP.

— Поддерживать подключение по WiFi (802.11a/b/g) и Ethernet (802.3i/y/ab: 10/100/1000 Мбит/с по витой паре).

— Поддерживать UART/RS485 либо по RX/TX/GND (5В).

— Иметь достаточно ресурсов для работы Web-сервера.

— Иметь возможность отладки при неисправности сетевого подключения.

Наиболее простым способом удовлетворить все требования является использование одноплатного микрокомпьютера с USB портами и/или поддерж-

кой UART: ОС микрокомпьютера обеспечивает поддержку сетевых подключений и предоставляет простой доступ к UART, для использования UART/RS485 можно задействовать USB переходник, либо же модифицировать ТРИД.

2.2 Подбор управляющего микрокомпьютера

В соответствие с 2.1 управляющий микрокомпьютер должен

— Помещаться в один корпус с ТРИД.

— Блок питания для управляющего микрокомпьютера должен помещаться там же.

— Иметь интерфейс USB (host), либо UART с RX/TX/GND и высоким уровнем 5 В.

— Иметь интерфейсы WiFi (IEEE 802.11a/b/g) и Ethernet (IEEE 802.3i/y/ab).

— Иметь достаточно ресурсов для работы Web-сервера.

— Иметь возможность отладки при неисправности сетевого подключения.

Помимо этого желательно наличие хорошей официальной службы поддержки либо сообщества людей с опытом разработки устройств на основе выбранного микрокомпьютера; второе предпочтительнее т.к. доказывает жизнеспособность систем на основе данного микрокомпьютера.

В качестве кандидатов в управляющие микрокомпьютеры был рассмотрен ряд различных одноплатных микрокомпьютеров: см. таблицу А.1. Исходя из требований наиболее подходящими были признаны Orange Pi Zero, Pine A64 и Raspberry Pi 3: первые два отличаются минимальной стоимостью при достаточном количестве ресурсов, последний имеет наиболее широкое сообщество.

Вследствие этого желательно, чтобы создаваемое ПО могло быть с минимальными затратами адаптировано для всех трёх микрокомпьютеров.

2.3 Выбор ОС и ПО

В соответствие с 2.1 операционная система должна соответствовать следующими требованиями:

— Работать на выбранном микрокомпьютере.

— Потреблять минимальные ресурсы на собственные нужды.

— Поддерживать контроллер UART и предоставлять к нему доступ.

— Поддерживать TCP/IP, иметь возможность написания собственного сервера.

— Поддерживать удалённый доступ для отладки сервера и загрузки ПО.

— Поддерживать все три выбранных микрокомпьютера.

Согласно официальному сайту Raspberry Pi[2] единственной официально поддерживаемой ОС является Raspbian, однако существует возможность установки альтернативных ОС. Raspbian является дистрибутивом GNU/Linux, основанном на Debian, что позволяет предположить, что ПО под этот дистрибутив будет с минимальными модификациями работать в других дистрибутивах, основанных на Debian. Так как все три выбранных микрокомпьютера поддерживают либо Debian (Pine A64, Orange Pi Zero), либо Raspbian (Raspberry Pi 3, Orange Pi Zero), то Raspbian удовлетворяет всем требованиям и необходимость лишаться официальной поддержки отсутствует.

Raspbian поставляется в двух вариантах: большой образ с PIXEL и минимальный образ без графического интерфейса[3]. В связи с тем, что для отладки возможно использование удалённого доступа по ssh, в минимальном образе также доступен framebuffer, служащий заменой графическому интерфейсу для отладки системы при физическом доступе, а основным способом доступа к управляемой системе был выбран Web-интерфейс, то необходимость использования большого образа отсутствует.

Далее, к управляемой системе предполагается доступ одного оператора с одного компьютера в единицу времени. Требуется также параллельное автоматизированное снятие показаний температуры из программы на языке LabView на том же компьютере. Вследствие этого была выбрана следующая архитектура: непосредственно с локальной сетью связан nginx, предоставляющий статические файлы для работы web интерфейса, а также служащий в качестве обратного прокси для специальной программы, предоставляющей доступ к ТРИД (см. рисунок 2.1). «Специальная программа» представляет собой Web-сервер с API, основанным на JSON и может быть использована из LabView без взаимодействия с остальной частью Web-интерфейса.

В указанной конфигурации JSON был выбран из-за широкой поддержки в различных языках программирования, в том числе в браузере. Nginx представляет собой широко используемый и лёгкий в настройке Web-сервер[4].

Для написания API backend был выбран язык Python: в число библиотек под этот язык входят библиотеки для работы с последовательным интерфейсом, библиотеки для создания Web сервисов, а также библиотеки для работы с протоколом modbus. Помимо этого, язык имеет безопасную модель памяти и обеспечивает быструю разработку за счёт быстродействия конечной программы. Для случая, если быстродействие недостаточно для нормальной работы, Python позволяет пере-

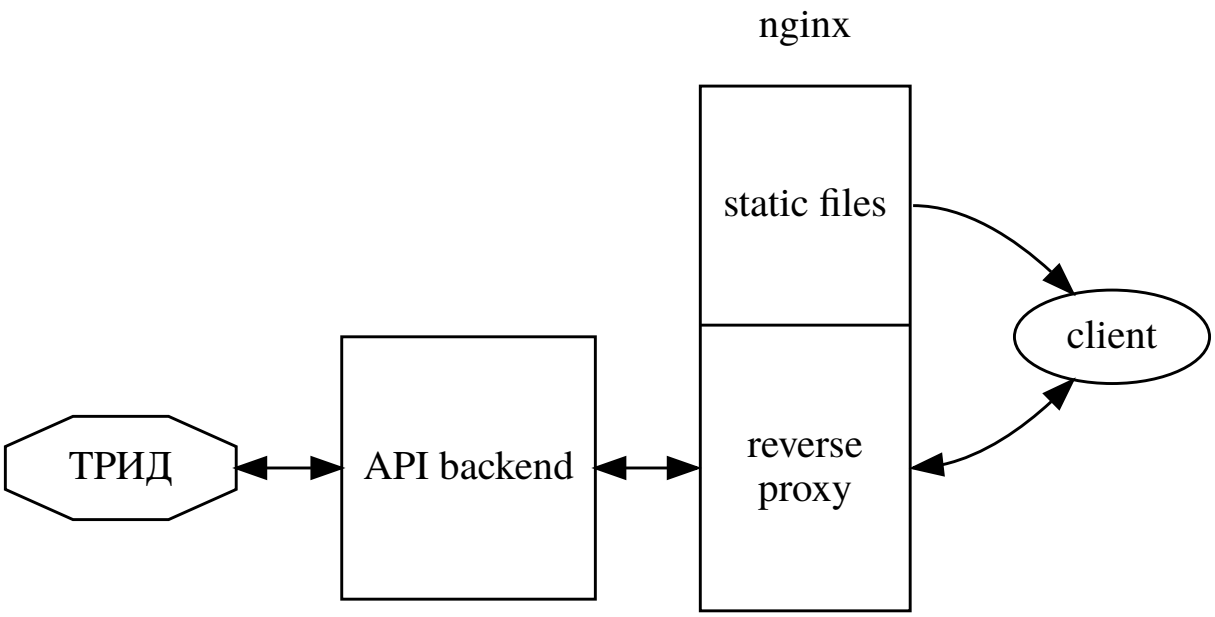

Рисунок 2.1 — Архитектура Web-сервиса

писывать часть программы на других языках, поддерживающих те же соглашения о вызове функций, что и C.

2.4 Выбор библиотек Python

Из предыдущего раздела видно, что для работы API backend требуются следующие библиотеки:

— Библиотека для работы с последовательным портом ввода-вывода.

— Библиотека для работы с протоколом modbus (поверх последовательного порта).

— Библиотека для сбора отладочной информации (создания журнала).

— Библиотека для создания Web сервиса.

В случае с библиотекой для работы с последовательным портом ввода-вывода есть фактически единственная альтернатива — pyserial[5], других развивающихся проектов данной тематики найти не удалось.

Протокол modbus поддерживается следующими библиотеками: MinimalModbus[6], pylibmodbus[7], modbus_tk[8], pymodbus[9], uModbus[10]. Из них pylibmodbus была отброшена из-за требований наличия дополнительной библиотеки на C, длительного отсутствия обновлений и отсутствия документации. Modbus_tk также отличалась отсутствием документации, а официальный пакет pymodbus в PyPI не поддерживается, хотя сама библиотека развивается[11]. Среди

оставшихся MinimalModbus и uModbus первая была выбрана за более удобный интерфейс.

В качестве библиотеки для сбора отладочной информации оказалось возможным использовать часть стандартной библиотеки Python, предназначеную для этой цели. На nginx была дополнительно возложена обязанность предоставления создаваемых данной библиотекой журналов.

Среди библиотек, подходящих для создания Web интерфейса обнаружилось наибольшее разнообразие. Библиотека circuits[12] была выбрана из-за того, что для её использования требуется написать минимальное количество кода, а также отсутствие требований к файловой структуре проекта.

3 Технологический раздел

3.1 Архитектура приложения

Как было показано в 2.3, приложение разбито на два основных модуля: nginx и API backend. В данном разделе будет рассмотрена только архитектура API backend, настройка nginx рассматривается отдельно в 3.2 а архитектура этой части тривиальна и фактически отображена на рисунке 2.1.

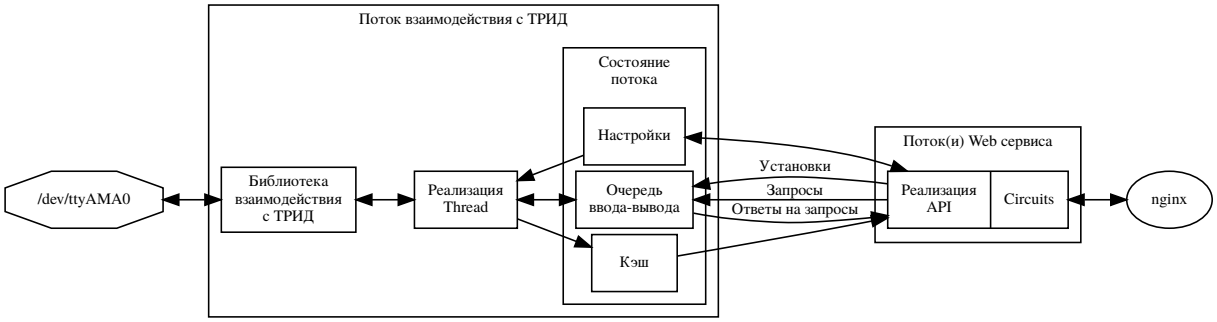

Рисунок 3.1 — Архитектура API backend

Архитектура API backend изображена на рис. 3.1. Всё приложение было разбито на три основных модуля: выделенная библиотека взаимодействия с ТРИД (используется внутри потока ввода-вывода), реализация потока ввода-вывода на основе классов из стандартной библиотеки Python — модуля **threading** и собственно реализации API на основе классов из библиотеки **circuits**.

Такая архитектура была создана из-за наличия только одной дуплексной линии связи с ТРИД: необходимо обеспечить строгую последовательность отправки запросов к ТРИД и получения ответов на случай прихода нового запроса от nginx во время обработки старого. Данную проблему можно решить, как минимум, на трёх уровнях: запретить nginx создавать более одного параллельного соединения к Web серверу, не использовать потоки при обработке запросов или работать с ТРИД исключительно через отдельный поток и очередь и/или блокировки. Третий вариант был выбран, из-за того, что существуют запросы, одновременная обработка которых безопасна. В основном это относится к запросам, получающим значения из кэша или настроек. Помимо этого поток взаимодействия с ТРИД требуется, чтобы постоянно опрашивать ТРИД, узнавая показания температурного датчика; без отдельного потока эту задачу пришлось бы возложить на Web-интерфейс, что привело бы к его замедлению.

3.2 Настройка nginx и circuits

В соответствие с рис. 2.1 nginx должен выполнять две функции: предоставлять доступ к статическим файлам (в них определяется весь Web-интерфейс) и предоставлять доступ к API backend. Для этого достаточно небольшой конфигурации, показанной в листинге 1.

```
server {
   listen 80 default_server;
   listen [::]:80 default_server;

   root /var/www/html;

   index index.html;

   server_name _;

   location / {
      # First attempt to serve request as file, then
      # as directory, then fall back to displaying a 404.
      try_files $uri $uri/ =404;
   }

   location /api {
      proxy_pass http://127.0.0.1:20000;
   }
}
```

Листинг 1 — Настройка Nginx

Данный файл настроек помещается в каталог **/etc/nginx/sites-enabled** под именем **default**.

В нём nginx настроен слушать на порту 80 (IPv4 и IPv6), брать статические файлы из /var/www/html и перенаправлять запросы с путём /api (к примеру, **http://mypihost/api/trid/monitor/get**) серверу (API backend) на порту 20 000. Для этого приложение Circuits должно выглядеть соответственно листингу 2.

```
from circuits.web import Controller, Server

class DummyAPI(Controller):
    '''Entry point for dummy API

     Handles requests to /api/dummy.

     :param dict common_state: Some common state.'''
    channel = '/api/dummy'

    def __init__(self, common_state, *args, **kwargs):
        super(DummyAPI, self).__init__(*args, **kwargs)
        self.__common_state = common_state
        common_state['dummy_counter'] = 0

    def increment(self):
        '''/api/dummy/increment request handler'''
        common_state['dummy_counter'] += 1
        return str(common_state['dummy_counter'])

class API(Controller):
    '''API entry point, serves nginx /api requests'''
    channel = '/api'
    '''Specify that this works with /api.'''

    def __init__(self, *args, **kwargs):
        super(API, self).__init__(*args, **kwargs)
        common_state = {}
        self += DummyAPI(common_state)


if __name__ == '__main__':
    application = Server(('127.0.0.1', 20000)) + API()
    aplication.run()
```

Листинг 2 — Настройка Circuits

Здесь создаётся простое приложение — Web-сервер, слушающий на порту 20 000 и предоставляющий всего одну возможность — узнавать, сколько раз был вызван API метод /**api**/**dummy**/**increment**. Порт, на котором сервер будет слушать сервер определяется в аргументах **Server**, атрибут **channel** служит для определения префикса метода(ов), которые будет обслуживать класс. Таким образом, /**api**/**dummy**/**increment** будет обслуживаться классом с атрибутом **channel**,

равным **/api/dummy** и имеющим метод **increment**, который и будет вызван при запросе **/api/dummy/increment**. **application.run()** в самом конце файла запускает бесконечный цикл, ждущий и обрабатывающий запросы.

3.3 Настройка автоматического запуска nginx и API backend

Одной из особенностей разрабатываемой системы являются частые перезагрузки, поэтому весьма желательно запускать nginx и API backend автоматически при старте системы. Для выполнения этой задачи можно использовать возможности дистрибутиве Raspbian, а именно систему инициализации systemd. Настройки, описывающие как система инициализации должна запускать nginx уже присутствуют в системе в качестве части пакета nginx, нужно только включить автозапуск: **sudo systemctl enable nginx**. Однако systemd не обладает информацией о способе запуска созданного API backend. Для исправления данной проблемы необходимо поместить в **/etc/systemd/system** файл **trid_site.service** с содержанием, указанным в листинге 3 и включить автозапуск: **sudo systemctl enable trid_site**.

```
1  [Unit]
2  Description=TRID API service
3  Wants=nginx.service
4  After=network.target
5
6  [Service]
7  Type=simple
8  ExecStart=/usr/bin/python /var/www/trid_site/main.py
9  PIDFile=/var/run/trid.pid
10 Restart=always
11 WorkingDirectory=/var/www
12
13 [Install]
14 WantedBy=multi-user.target
```

Листинг 3 — Настройка systemd

Здесь указывается, что сервис (API backend) требует для работы nginx (**Wants=nginx.service**), а для работы сервиса нужно просто запустить **/usr/bin/python /var/www/trid_site/main.py** в каталоге **/var/www**. Образец того, как должен выглядеть **/var/www/trid_site/main.py** есть в листинге 2.

3.4 Создание библиотеки взаимодействия с ТРИД

Библиотека взаимодействия с ТРИД абстрагирует низкоуровневый доступ к ТРИД, скрывая детали реализации протокола modbus, используемые адреса регистров и разбор ответа от ТРИД. Библиотека также позволяет проверить корректность введённых пользователем данных. Однако соответствие значениям из табл. Г.4 не проверяется: за попыткой ввода значений, выходящих за данные пределы, не следует отказ ТРИД, поэтому эти значения сохранены только в Web-интерфейсе.

Библиотека предоставляет один класс для доступа к ТРИД (названный **TRID**) и набор констант для проверки пользовательского ввода (константы, начинающиеся с **SUPPORTED_**), а также набор констант со значениями по-умолчанию (константы, начинающиеся с **DEFAULT_**). Библиотека зависит только от стандартной библиотеки Python, pyserial[5] и MinimalModbus[6]. Сама библиотека называется **trid**.

Полное определение API библиотеки есть в приложении Б.

3.4.1 Настройка последовательного порта

Библиотека MinimalModbus предполагает использование pyserial для доступа к порту, а именно являющегося частью библиотеки класса **serial.Serial**. Однако API MinimalModbus предполагает, что при инициализации класса **minimalmodbus.Instrument** будет передан единственный аргумент **serial.Serial** — **port**, то есть путь к последовательному порту (к примеру, **/dev/ttyAMA0**). Дополнительные настройки (скорость, чётность, размер слова, ...) необходимо устанавливать, используя атрибут **Instrument.serial**.

Помимо этого, ТРИД предоставляет ограниченное количество комбинаций чётности, числа конечных бит и размера одного слова, из-за чего эти три настройки были объединены в один класс **trid.PortSettings**, а возможные комбинации сохранены в **trid.SUPPORTED_PORT_SETTINGS**. Таким образом, код настройки соединения с ТРИД, был прописан в методе инициализации класса и представлен в листинге 4.

```
class TRID(object):
   '''PID regulator interface

    May be used like this: ``with TRID('/dev/ttyAMA0') as t: ...``, this will
    automatically close /dev/ttyAMA0 at __exit__. It may also be closed with
    :py:meth:`TRID.close`.

    :param str tty: Terminal to work with.
    :param str mode: Mode: 'asc' or 'rtu'.
    :param int slaveaddress: PID regulator address. May be from 0x01 to 0xFF.
    :param int baudrate: PID regulator baud rate. Must be one of the values in
        SUPPORTED_BAUDRATES.
    :param PortSettings port_settings: PID regulator port settings, see
        :py:class:`PortSettings`.
    :param float timeout: Maximum time to wait when reading, in seconds.'''
   def __init__(self, port, mode=DEFAULT_MODE,
     slaveaddress=DEFAULT_SLAVE_ADDRESS, baudrate=DEFAULT_BAUD_RATE,
     port_settings=DEFAULT_PORT_SETTINGS, timeout=DEFAULT_TIMEOUT):

     assert(mode in MODE_TRANSLATIONS)
     assert(0x01 <= slaveaddress <= 0xFF)
     assert(baudrate in SUPPORTED_BAUDRATES)
     assert(port_settings in SUPPORTED_PORT_SETTINGS)
     self.mode = mode
     '''Currently effective mode'''
     self.slaveaddress = slaveaddress
     '''Currently used slave address'''
     self._instrument = Instrument(port, slaveaddress,
                                   MODE_TRANSLATIONS[mode])
     '''MinimalModbus interface.'''
     instr = self._instrument
     instr.serial.baudrate = baudrate
     instr.serial.parity = port_settings.parity
     instr.serial.bytesize = port_settings.bytesize
     instr.serial.stopbits = port_settings.stopbits
     instr.serial.timeout = timeout
```

Листинг 4 — Код инициализации класс TRID

3.4.2 Протокол взаимодействия с ТРИД

Согласно инструкции, ТРИД использует протокол modbus, однако некоторые детали не описаны в инструкции[1]. Управление ТРИД осуществляется путём

установки 16-битных регистров, перечисленных на стр. 22. Значения в данных регистрах предоставляют из себя 16-битные знаковые целые, представленные в виде дополненного кода. Некоторые из этих регистров являются целыми числами, полученными из рациональных путём умножения на 0,1, в таблице это отражено в столбце «единицы измерения» (к примеру, 0,1 °C). При запросе регистров допустимо запрашивать два регистра одновременно, если их адрес отличается не более, чем на единицу. Таким свойством обладают только регистры одинакового назначения, но для разных каналов: к примеру, «измеренное значение, канал 1» по адресу 0x0000 и «измеренное значение, канал 2» по адресу 0x0001.

При использовании MinimalModbus для реализации данного протокола достаточно четырёх вспомогательных функций: двух для конвертации 16-битных целых в/из двоичного кода, показанных в листинге 5 и двух методов класса.

```
def from_twos_complement(u16):
    '''Convert 16-bit unsigned integer to 16-bit signed integer'''
    mask = 2 ** (16 - 1)
    return -(u16 & mask) + (u16 & ~mask)

def to_twos_complement(i16):
    '''Convert 16-bit signed integer to 16-bit unsigned integer'''
    if i16 >= 0:
        return i16
    else:
        mask = (2 ** 16) - 1
        return (-1 - i16) ^ mask
```

Листинг 5 — Функции конвертации знаковых 16-битных целых в/из 16-битных беззнаковых целых чисел

Функции конвертации 16-битных чисел предполагают, что MinimalModbus воспринимает значения как 16-битные беззнаковые целые, сохранённые в стандартном типе **int**, тогда как входные значения представляют из себя 16-битные знаковые целые, также сохранённые в этом типе, однако не предполагая определённого формата их представления. При этом стандартный тип **int** представляет из себя большое целое число, поддерживающий битовые операции.

```
class TRID(object):
    # …
    def _get_ch_regs(self, address, channel, decimals=1):
        assert(channel in {None, 1, 2})
        self._instrument.serial.flushInput()
        if channel is None:
            ch1, ch2 = self._instrument.read_registers(address, 2)
            return (from_twos_complement(ch1) / (10 ** decimals),
                    from_twos_complement(ch2) / (10 ** decimals))
        else:
            address = address + channel - 1
            return from_twos_complement(self._instrument.read_register(
                address + channel - 1)) / (10 ** decimals)

    def _set_ch_regs(self, values, address, channel, decimals=1):
        assert(channel in {None, 1, 2})
        if channel is None:
            v1, v2 = (to_twos_complement(int(v * (10 ** decimals)))
                      for v in values)
            v = [v1, v2]
            return self._instrument.write_registers(address, v)
        else:
            address = address + channel - 1
            return self._instrument.write_register(
                address, to_twos_complement(values * (10 ** decimals)))
```

Листинг 6 — Реализация протокола взаимодействия с ТРИД на основе MinimalModbus

В листинге 6 показаны приватные методы класса **TRID**, используемые для реализации публичных методов. Вследствие их использования все публичные методы класса **TRID** состоят из единственного вызова одного из приватных методов с правильным адресом регистра, значением decimals (определяющим, сколько раз 16-битное целое нужно разделить на 10 для получения правильного значения: в большинстве случаев один раз, в остальных — 0) и предоставленными пользователем остальными аргументами.

3.5 Поток взаимодействия с ТРИД

Поток взаимодействия с ТРИД отвечает за синхронную обработку запросов к ТРИД, а именно, запросов на получение текущих значений настроек и их установку, а также за периодический опрос ТРИД для получения показаний тем-

пературных датчиков. Поток разделён на три основные части: библиотеку взаимодействия с ТРИД, описанную в 3.4, класс состояния потока, и собственно сама реализация потока на основе класса **threading.Thread** из стандартной библиотеки Python.

3.5.1 Класс состояния потока

Класс состояния потока содержит

а) очередь заданий (**Queue.Queue** из стандартной библиотеки), в которую поступают все запросы на взаимодействие с ТРИД, за исключением периодического опроса ТРИД и запросов на установку целевой температуры,

б) экземпляр класса **trid.TRID**, являющийся интерфейсом библиотеки взаимодействия,

в) информацию о целевой температуре, включая состояния «нагрев включён/отключён» раздельно для обоих каналов,

г) настройки опроса: интервал опроса и максимальное количество сохраняемых значений,

д) собственно сохранённые показания температурных датчиков,

е) блокировки:

1) совместная/исключительная блокировка для предотвращения получения несогласованных данных при запросе показаний датчиков (реализация блокировки взята из [13]),

2) исключительная блокировка (**threading.Lock** из стандартной библиотеки), предотвращающая переподключение к ТРИД во время его опроса,

3) исключительная блокировка (**threading.Lock**) для предотвращения получения несогласованных установок (настроек опроса и целевой температуры);

ж) событие (**threading.Event** из стандартной библиотеки) остановки, позволяющее мягко завершить поток,

з) вспомогательные методы для подключения к ТРИД, сохранения настроек, которые также сохраняются в долговременую память и установки/получения целевой температуры.

Основные задачи класса — хранение состояния потока, через которое также обеспечивается взаимодействие с реализацией API на основе circuits. Методы класса не абстрагируют использование блокировок, взаимодействие с очередью и событием или получение данных.

Метод инициализации класса приведён в листинге 7, вспомогательные функции — в листинге 8.

```
import threading
import Queue
class PIDControllerThreadState(object):
   def __init__(self, port_name='pi-uart'):
      self.shutdown_event = threading.Event()
      self.job_queue = Queue.Queue()
      self.data_lock = RWLock()
      '''RWLock used for exported data'''
      self.data = PIDControllerData()
      '''Data read/written to by the thread'''

      self.trid = None
      self.trid_port = port_name
      self.trid_lock = threading.Lock()

      self.interval = 1.0
      self.data_points = 100
      self.target_enabled = (None, None)
      self.target_temp = (None, None)
      '''Last set target temperature'''
      self.did_set_target_temp = False
      '''True if setting target temperature is not needed'''
      self.need_save_settings = True
      '''True if saving settings is needed'''
      self.state_lock = threading.Lock()

      self.trid_connect(port_name)
      self.save_settings()
```

Листинг 7 — Метод инициализации состояния потока

```
DISABLED_TEMP = -200
'''Temperature set when disabling heating.'''

class PIDControllerThreadState(object):
   # …
   def trid_connect(self, port_name, **kwargs):
      '''(Re)connect to PID regulator

         Keyword arguments are passed directly to :py:class:`trid.TRID`.

```

```
11          :param str port_name: Port name, key in PORTS global.'''
12      target_temp = (None, None)
13      try:
14          with open(SAVED_SETTINGS_FILE, 'r') as fp:
15              settings = json.load(fp)
16      except IOError:
17          pass
18      else:
19          with self.state_lock:
20              target_temp = tuple(settings['target_temp'])
21              self.interval = settings['interval']
22              self.data_points = settings['data_points']
23      with self.trid_lock:
24          if self.trid is not None:
25              self.trid.close()
26          self.trid = trid.TRID(PORTS[port_name], **kwargs)
27      with self.state_lock:
28          self.set_target_temp(target_temp, (False, False))
29      self.save_settings()
30
31  def save_settings(self):
32      '''Save settings: target temperature, interval, update time
33
34          Saves to SAVED_SETTINGS_FILE.'''
35      if not self.need_save_settings:
36          return
37      with self.state_lock:
38          settings = {
39              'target_temp': self.target_temp,
40              'interval': self.interval,
41              'data_points': self.data_points,
42          }
43          self.need_save_settings = False
44      try:
45          with open(SAVED_SETTINGS_FILE, 'w') as fp:
46              json.dump(settings, fp)
47      except Exception:
48          logger.exception('Failed to save settings')
49          self.need_save_settings = True
50
51  def set_target_temp(self, target_temp, target_enabled):
52      '''Set target temperature and record it in the file
53
54          Must be called under :py:attr:`state_lock`.
55
```

```
56              File to record in: SAVED_SETTINGS_FILE.'''
57          target_temp = (
58              first_non_none(target_temp[0], self.target_temp[0], DISABLED_TEMP),
59              first_non_none(target_temp[1], self.target_temp[1], DISABLED_TEMP),
60          )
61          target_enabled = (
62              first_non_none(target_enabled[0], self.target_enabled[0], False),
63              first_non_none(target_enabled[1], self.target_enabled[1], False),
64          )
65          self.target_enabled = target_enabled
66          self.target_temp = target_temp
67          self.need_save_settings = True
68          self.did_set_target_temp = False
69
70      def get_target_temp(self):
71          '''Get effective target temp (what will be transferred to TRID)
72
73              Must be called under :py:attr:`state_lock`.'''
74          target_temp = self.target_temp
75          enabled = self.target_enabled
76          return (
77              (target_temp[0] if enabled[0] else DISABLED_TEMP),
78              (target_temp[1] if enabled[1] else DISABLED_TEMP),
79          )
```

Листинг 8 — Вспомогательные функции класса состояния потока

3.5.2 Реализация потока взаимодействия с ТРИД

Основной частью реализации потока взаимодействия с ТРИД является потомок класса **threading.Thread** из стандартной библиотеки Python. Именно в этой части реализован периодический опрос и синхронная обработка заданий из очереди.

Поток взаимодействия с ТРИД действует следующим образом:

а) Получает настройки от класса состояния (рис. В.1).

б) Если в соответствии с настройками требуется установка целевой температуры ТРИД, устанавливает целевую температуру (рис. В.2).

в) Получает и сохраняет показания температурных датчиков (рис. В.3).

г) Обрабатывает задачи, полученные от оператора (рис. В.5).

Задача от оператора обрабатывается, если она либо уже есть в очереди, либо если она пришла за время, не превышающее интервал между опросами ТРИД с учётом того, что предыдущие шаги уже заняли некоторое количество времени.

При наличии задач от оператора, но отсутствии времени (к примеру, если интервал между опросами установлен в ноль), обрабатывается ровно одна задача от оператора на одно считывание показаний температурных датчиков.

Подробная блок-схема потока приведена в прил. В, код потока есть в листинге 14.

3.6 Создание API

В соответствие с табл. Б.3, по UART доступны следующие возможности ТРИД: получение/установка ширины гистерезиса, получение/установка значений аварийной температуры (3 канала), установка коэффициентов для ПИД-регулирования (K_p, K_i, K_d), получение/установка целевой температуры, получение показаний температурных датчиков. В дополнении к этому API поддерживает получение показаний температуры за данный период (все сохранённые показания, либо показания температуры, снятые после указанного времени), сброс сохранённых показаний, установку параметров подключения к ТРИД, установку максимального числа хранимых показаний температуры и частоты опроса ТРИД.

Для облегчения работы с ТРИД API получения/установки целевой температуры поддерживает состояния «нагрев отключён» (ТРИД в этом случае используется только для снятия показаний) и «нагрев включён». Данные состояния реализуются через указание минимальной (-200 °C) температуры в качестве целевой, таким образом не предполагая использования ТРИД с подключением к охлаждающему устройству помимо нагревающего. В табл. Г.1 перечислены все API методы.

Также API backend сохраняет информацию о параметрах подключения, параметрах снятия показаний (интервал, сохранённое количество) и последней использованной целевой температуре в файловой системе управляющего микрокомпьютера. Независимо от сохранённой целевой температуры при запуске сервиса включается режим «нагрев отключён».

В ходе проверки разработанного приложения было выяснено, что использование каналов аварийной сигнализации, за исключением канала A, ведёт к отказу ТРИД: в зависимости от способа использования ТРИД либо немедленно перезагружается, либо пишет в канал связи данные, не соответствующие протоколу modbus. Поэтому API сервиса не поддерживает установку каналов аварийной сигнализации, отличных от A.

3.6.1 Описание API на основе классов circuits

Как несложно увидеть, значительная часть методов из таблицы Г.1 представляет из себя просто пару из получения метода и его установки. Все данные методы реализуются на основе общего предка **TRIDGetSetAPI**, показанного в листинге 9.

```
import circuits.web.exceptions as cwe
from circuits.web import Controller

class TRIDGetSetAPI(Controller):
   '''/api/*/{get,set} API entry point

     :param PIDControllerThreadState thread_state:
         PID controller thread state.'''
   def __init__(self, thread_state, *args, **kwargs):
      super(TRIDGetSetAPI, self).__init__(*args, **kwargs)
      self.__thread_state = thread_state

   def get(self, channel=None):
      if channel not in {None, '1', '2'}:
         raise cwe.BadRequest(json.dumps({'error': 'Unexpected channel'}))
      return request(self.__thread_state, 'get_' + self._trid_suffix,
                     channel, **self._trid_kwargs)

   def POST(self, value1=None, value2=None):
      if value1 is None and value2 is None:
         raise cwe.BadRequest(json.dumps({
            'error': 'Must supply at least one value'}))
      try:
         if value1 is not None: value1 = float(value1)
         if value2 is not None: value2 = float(value2)
      except ValueError:
         raise cwe.BadRequest(json.dumps({
            'error': 'Must use floating-point values as values'}))
      if   value1 is None: channel = 2
      elif value2 is None: channel = 1
      else:                channel = None
      values = (value1, value2)
      if channel is not None:
         values = values[channel - 1]
      return request(self.__thread_state, 'set_' + self._trid_suffix,
                     channel, values=values, **self._trid_kwargs)
```

Листинг 9 — Общий класс-предок для большей части API, взаимодействующей с ТРИД

В соответствии с API **circuits** метод **POST** указанного класса вызывается при POST-запросе по пути, указанном в атрибуте **channel**. Метод **get**, как и любой другой метод, начинающийся с маленькой буквы и содержащий только

маленькие буквы и знаки подчёркивания[1] вызывается при GET-запросе по адресу **{channel}/{method_name}**, где **{channel}** — путь, содержащийся в атрибуте **channel**, а **{method_name}** — имя метода (в данном случае — **get**). Если всё в порядке, то оба рассматриваемых метода возвращают строку, см. листинг 11. В случае ошибок возбуждаются исключения из модуля **circuits.web.exceptions**, преобразующиеся в соответствующие HTTP коды состояния[14, секция 6].

```
1  class PIDCoefKpGetSetAPI(TRIDGetSetAPI):
2      '''/api/coef/kp/* API entry point
3
4       :param PIDControllerThreadState thread_state:
5           PID controller thread state.'''
6      channel = '/api/coef/kp'
7      _trid_suffix = 'pid_coef'
8      _trid_kwargs = {'coef': 'Kp'}
```

Листинг 10 — Пример класса-потомка **TRIDGetSetAPI**

Помимо атрибута **channel** ожидается, что классы-потомки **TRIDGetSetAPI** установят атрибуты **_trid_suffix** и **_trid_kwargs**. Первый атрибут — строка, указывающая метод **trid.TRID** для запроса в функции **request** (используются два метода: **get_{_trid_suffix}** и **set_{_trid_suffix}** для **get** и **POST** соответственно); второй описывает дополнительные аргументы для запрашиваемого метода. Пример класса-потомка приведён в листинге 10, код всех используемых таких классов приведён в листинге 15.

[1] Всё только из таблицы ASCII символов.

```
import json
import threading
import circuits.web.exceptions as cwe

REQUEST_WAIT_TIMEOUT = 10

def request(thread_state, meth, channel=None, **kwargs):
    '''Request register value from the PID regulator

      :param str meth: :py:class:`trid.TRID` method name.
      :param channel: Channel: 1, 2 or ``None``.
      :param kwargs: Additional arguments, if needed.

      :return: Request body. May raise exceptions from
          :py:module:`circuits.web.exceptions`.'''
    finish_evt = threading.Event()
    kw = {'channel': channel if channel is None else int(channel)}
    kw.update(kwargs)
    response = []
    thread_state.job_queue.put((finish_evt, meth, kw, response))
    if finish_evt.wait(REQUEST_WAIT_TIMEOUT):
        err, value = response[0]
        if not err:
            raise cwe.InternalServerError(*value)
        if value is not None:
            if channel is None:
                value = {'value1': value[0], 'value2': value[1]}
            else:
                value = {'value' + str(channel): value}
        return json.dumps(value)
    else:
        raise cwe.RequestTimeout()
```

Листинг 11 — Вспомогательная функция **request**

В листинге 11 показано, как происходит общение с потоком взаимодействия с ТРИД: в очередь событий помещается событие, создаваемое на месте и затем устанавливаемое в потоке взаимодействия по завершению обработки запроса, имя метода класса **trid.TRID**, аргументы этого метода и контейнер для возврата результата. Ожидается, что после установки события в контейнере окажется пара с двумя значениями: флаг ошибки (ложен, если ошибка, истинен в её отсутсвие) и собственно ответ — описание ошибки в виде списка аргументов

для **circuits.web.exceptions.InternalServerError** либо значение, возвращённое запрошенным методом **trid.TRID**. Возвращённое значение затем преобразуется в строку формата JSON[15]. Для того, чтобы предотвратить бесконечное ожидание в случае программной ошибки ожидание установки события ведётся не более **REQUEST_WAIT_TIMEOUT** (т.е. десяти) секунд, затем возбуждается исключение. Для упрощения кода и вследствие того, что известно, что методы **trid.TRID.set_*** возвращают исключительно **None**, соглашение «POST методы должны вернуть **null** при отсутствии ошибки», прослеживаемое в табл. Г.1 не обрабатывается отдельно: **json.dumps(None)** возвращает **null**.

Вследствие наличия состояния «нагрев включён/отключён», а также того, что целевая температура в сервисе является настройкой потока взаимодействия с ТРИД, а не параметром, запрашиваемым у ТРИД, несмотря на внешную схожесть методов для **/api/target** не используется наследование от **TRIDGetSetAPI**: код класса **TargetGetSetAPI**, обрабатывающего приведён в листинге 12.

```
1  import circuits.web.exceptions as cwe
2  from circuits.web import Controller
3
4  DISABLED_TEMP = -200
5
6  class TargetGetSetAPI(Controller):
7      channel = '/api/target'
8      def __init__(self, thread_state, *args, **kwargs):
9          super(TargetGetSetAPI, self).__init__(*args, **kwargs)
10         self.__thread_state = thread_state
11     def get(self, channel=None):
12         if channel not in {None, '1', '2'}:
13             raise cwe.BadRequest(json.dumps({'error': 'Unexpected channel'}))
14         ret = {}
15         with self.__thread_state.state_lock:
16             ret['enabled1'], ret['enabled2'] = (
17                 self.__thread_state.target_enabled)
18         if ret['enabled1'] or ret['enabled2']:
19             request(self.__thread_state, 'get_target_temp', channel)
20         with self.__thread_state.state_lock:
21             ret['value1'], ret['value2'] = (
22                 self.__thread_state.target_temp)
23         return json.dumps(ret)
24     def POST(self, value1=None, value2=None, enabled1=None, enabled2=None):
25         if (value1 is None and value2 is None
26             and enabled1 is None and enabled2 is None):
```

```
27            raise cwe.BadRequest(json.dumps({
28                'error': 'Must supply at least one value'}))
29        try:
30            if value1 is not None: value1 = float(value1)
31            if value2 is not None: value2 = float(value2)
32        except ValueError:
33            raise cwe.BadRequest(json.dumps({
34                'error': 'Must use floating-point values as values'}))
35        try:
36            if enabled1 is not None: enabled1 = to_boolean(enabled1)
37            if enabled2 is not None: enabled2 = to_boolean(enabled2)
38        except ValueError:
39            raise cwe.BadRequest(json.dumps({
40                'error': 'Must use boolean values as enabled switches'}))
41        with self.__thread_state.state_lock:
42            target_temp = list(self.__thread_state.target_temp)
43            if value1 is not None: target_temp[0] = value1
44            if value2 is not None: target_temp[1] = value2
45            enabled = list(self.__thread_state.target_enabled)
46            if enabled1 is not None: enabled[0] = enabled1
47            if enabled2 is not None: enabled[1] = enabled2
48            self.__thread_state.set_target_temp(target_temp, enabled)
49        return 'null'
```

Листинг 12 — Код класса **TargetGetSetAPI**, обрабатывающего запросы к **/api/target**

Как видно на листинге, запрос текущей целевой температуры обрабатывается путём получения настроек (атрибутов **PIDControllerThreadState**) **target_temp** (значения температуры) и **target_enabled** (состояния флага отключённости нагрева), защищённого блокировкой **state_lock**. Установка целевой температуры осуществляется путём установки этих же атрибутов и ещё двух под той же блокировкой путём вызова метода **set_target_temp**. Два дополнительных атрибута — это **did_set_target_temp** для указания потоку взаимодействия с ТРИД на необходимость установки целевой температуры у ТРИД и **need_save_settings** для указания ему же на необходимость сохранения изменённых настроек (см. листинг 8).

Получение сохранённых показаний температурных датчиков происходит схожим с получением текущей целевой температуры образом, только вместо **state_lock** используется совместная блокировка **data_lock**. В потоке взаимодействия с ТРИД эта же блокировка используется в исключительном режиме. Для отбрасывания старых показаний (т.е. полученных до временной отметки из пара-

метра **timestamp**) используется модуль **bisect** из стандартной библиотеки Python, позволяющий в одну строку осуществлять поиск методом дихотомии. Код соответствующего класса приведён в листинге 16. Обработка сброса показаний осуществляется отдельно в классе **TemperatureClearAPI** (см. листинг 17).

Помимо этого API сервиса также предоставляет доступ к настройкам подключения ТРИД и настройкам опроса температурных датчиков: **/api/trid/connect** и **/api/trid/monitor**. Код классов-обработчиков данных запросов приведён в листингах 18 и 19 соответственно.

3.6.2 Основная точка входа в сервис

Как видно из 3.6.1 все классы-обработчики API запросов для инициализации требуют параметр **thread_state**. Помимо этого есть желание сделать так, чтобы код запуска сервиса был бы так же прост, как и в листинге 2. Для этого все классы, описанные в 3.6.1 инициализируются в одном классе-точке входа **API**, приведённом в листинге 13.

```
from circuits.web import Controller
class API(Controller):
    '''API entry point, serves nginx /api requests'''
    channel = '/api'
    '''Specify that this works with /api.'''

    def __init__(self, *args, **kwargs):
        super(API, self).__init__(*args, **kwargs)
        self.__thread_state = PIDControllerThreadState()
        self.__thread = PIDControllerThread(self.__thread_state)
        self.__thread.start()
        self += TemperatureAPI(self.__thread_state)
        self += TRIDConnectAPI(self.__thread_state)
        self += TargetGetSetAPI(self.__thread_state)
        self += HysteresisGetSetAPI(self.__thread_state)
        self += AlarmAGetSetAPI(self.__thread_state)
        self += PIDCoefKpGetSetAPI(self.__thread_state)
        self += PIDCoefKiGetSetAPI(self.__thread_state)
        self += PIDCoefKdGetSetAPI(self.__thread_state)
        self += TRIDMonitorAPI(self.__thread_state)
```

Листинг 13 — Класс-точка входа в сервис

Указанный класс является ответственным за инициализацию всех используемых классов-обработчиков запросов, а также запуск потока взаимодействия

с ТРИД с инициализацией состояния потока. С его использованием последняя часть приложения (начиная со строки **if __name__ == '__main__'**) выглядит точно так же, как и в листинге 2.

4 Экспериментальный раздел

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ООО «Вектор-ПМ», Пермь. — ПИД-регулятор температуры двухканальный ТРИД РТП322, Руководство по эксплуатации ВПМ 421210.009-18 РЭ, 2012.

2. Raspberry Pi Downloads. — 2017. — February. — Access mode: https://www.raspberrypi.org/downloads/.

3. Raspberry Pi Downloads. — 2017. — February. — Access mode: https://www.raspberrypi.org/downloads/raspbian/.

4. Historical trends in the usage of web servers for websites. — 2017. — February. — Access mode: https://w3techs.com/technologies/history_overview/web_server.

5. pyserial. — 2017. — March. — Access mode: https://pypi.python.org/pypi/pyserial.

6. MinimalModbus. — 2017. — March. — Access mode: https://pypi.python.org/pypi/MinimalModbus.

7. pylibmodbus. — 2017. — March. — Access mode: https://pypi.python.org/pypi/pylibmodbus.

8. modbus_tk. — 2017. — March. — Access mode: https://pypi.python.org/pypi/modbus_tk.

9. pymodbus. — 2017. — March. — Access mode: https://pypi.python.org/pypi/pymodbus.

10. uModbus. — 2017. — March. — Access mode: https://pypi.python.org/pypi/uModbus.

11. pymodbus github repository. — 2017. — March. — Access mode: https://github.com/bashwork/pymodbus.

12. circuits. — 2017. — March. — Access mode: https://pypi.python.org/pypi/circuits.

13. Python : Any way to get one process to have a write lock and others to just read on parallel? — 2017. — April. — Access mode: http://stackoverflow.com/questions/16261902.

14. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content : RFC : 7231 / RFC Editor ; Executor: Julian F. Reschke Roy T. Fielding : 2014. — June. — P. 101. Access mode: https://tools.ietf.org/html/rfc7231.

15. The JavaScript Object Notation (JSON) Data Interchange Format : RFC : 7159 / RFC Editor ; Executor: Tim Bray : 2014. — March. — P. 16. Access mode: https://tools.ietf.org/html/rfc7159.

ПРИЛОЖЕНИЕ А Сравнительные характеристики микрокомпьютеров

Таблица А.1 — Сравнительные характеристики микрокомпьютеров

Название микрокомпьютера	Цена, руб.	USB, UART	WiFi, 802.11x	Ethernet, xBASE-T	ОЗУ, МиБ	Долговременная память, макс.	Размеры В×Ш, мм
I	II	III	IV	V	VI	VII	VIII
ODROID-C2	4 500	USB, UART	через USB	10/100/1000	2 048	MicroSD/eMMC, 64 ГиБ	85×56
Pine A64+	1 700	USB, UART	b/g/n	10/100/1000	2 048	MicroSD, 256 ГиБ	133×80
Pine A64	1 000	USB, UART	b/g/n	10/100	512	MicroSD, 256 ГиБ	133×80
BeagleBone Black Rev C	3 400	USB, UART	b/g/n[1]	10/100	512	eMMC, 4 ГиБ	88×55
Banana Pi BPI-M1+	2 400	USB, UART	b/g/n	10/100/1000	1 024	MicroSD/SATA, 2 ТиБ	92×60
Intel Galileo Gen2	2 900	USB, UART	через USB	10/100	256	MicroSD, 32 ГиБ	124×72
Orange Pi Zero	1 000	USB, UART	b/g/n	10/100	256	TF/MMC, 64 ГиБ	48×46
Raspberry Pi 3	2 400	USB, UART	n	10/100	1 024	MicroSD, 64 ГиБ	85×56
MB77.07	4 800	USB, UART	через USB	10/100	256	встроенная	80×80

[1]Вместо Ethernet: микрокомпьютер может поставляться с Ethernet или WiFi, но не и с тем, и с другим.

Таблица Б.1 — Константы со значениями по-умолчанию

Название	Значение	Описание
DEFAULT_MODE	asc	Режим протокола: текстовый.
DEFAULT_BAUD_RATE	9 600	Скорость обмена данными.
DEFAULT_SLAVE_ADDRESS	1	Адрес ТРИД.
DEFAULT_PORT_SETTINGS	**(PARITY_NONE, STOPITS_ONE, EIGHTBITS)**	Настройки порта: без битов чётности, с одним стоповым битом и 8-битным словом.
DEFAULT_TIMEOUT	0.5	Время ожидания ответа от ТРИД в секундах.

Таблица Б.2 — Константы со поддерживаемыми значениями (словари или множества)

Название	Описание
SUPPORTED_BAUDRATES	Множество поддерживаемых скоростей обмена данными.
SUPPORTED_PORT_SETTINGS	Множество поддерживаемых настроек порта (кортежи с чётностью, числом стоповых бит и размером слова).
MODE_TRANSLATIONS	Словарь с поддерживаемыми режимами modbus (текстовым/бинарным).

Таблица Б.3 — Методы класса TRID

Метод	Описание	Аргументы	Возвращаемое значение
I	II	III	IV
read_temperature	Получить показания температурных датчиков.	**channel** — канал: 1, 2 или **None** для получения с обоих каналов.	Одно значение (число), если канал 1 или 2, иначе кортеж с двумя значениями.
get_target_temp	Получить установленную целевую температуру.	**channel** — канал: 1, 2 или **None** для получения с обоих каналов.	Одно значение (число), если канал 1 или 2, иначе кортеж с двумя значениями.
get_alarm	Получить установленное значение аварийной температуры.	**channel** — канал: 1, 2 или **None** для получения с обоих каналов; **alarm** — канал аварийной сигнализации: A, B или C.	Одно значение (число), если канал 1 или 2, иначе кортеж с двумя значениями.
get_hyst	Получить значение гистерезиса.	**channel** — канал: 1, 2 или **None** для получения с обоих каналов.	Одно значение (число), если канал 1 или 2, иначе кортеж с двумя значениями.
get_pid_coef	Получить значение ПИД коэффициента.	**channel** — канал: 1, 2 или **None** для получения с обоих каналов; **coef** — коэффициент: Kp, Ki или Kd.	Одно значение (число), если канал 1 или 2, иначе кортеж с двумя значениями.

I	II	III	IV
set_target_temp	Установить значение целевой температуры.	**channel** — канал: 1, 2 или **None** для установки на обоих каналах; **values** — одно числовое значение, либо кортеж с двумя.	нет
set_alarm	Установить аварийную сигнализацию.	**channel** — канал: 1, 2 или **None** для установки на обоих каналах; **values** — одно числовое значение, либо кортеж с двумя; **alarm** — канал аварийной сигнализации: A, B или C.	нет
set_hyst	Установить значение гистерезиса.	**channel** — канал: 1, 2 или **None** для установки на обоих каналах; **values** — одно числовое значение, либо кортеж с двумя.	нет
set_pid_coef	Установить значение коэффициента ПИД-регулирования.	**channel** — канал: 1, 2 или **None** для установки на обоих каналах; **values** — одно числовое значение, либо кортеж с двумя; **coef** — коэффициент: Kp, Ki или Kd.	нет

ПРИЛОЖЕНИЕ В Реализация потока взаимодействия с ТРИД

Таблица В.1 — Символы блок-схемы

Символ в блок-схеме	Тип символа	Тип значения	Описание
I	II	III	IV
e	Событие	-	Выход из цикла
t	Переменная	Рац. число	Время начала итерации
S	Блокировка	-	Блокировка настроек
интервал	Настройка	Неотриц. число	Интервал опроса ТРИД
i	Переменная	Неотриц. число	Сохранённая настройка: интервал опроса ТРИД
p	Переменная	Нат. число	Сохранённая настройка: число сохраняемых значений температуры
s	Переменная	Булево значение	Флаг: требуется ли установка целевой температуры
g	Переменная	Два рац. числа	Сохранённая настройка: целевая температура для каналов
L	Блокировка	-	Блокировка доступа к ТРИД
d	Переменная	Два рац. числа	Показания температурных датчиков ТРИД (два канала)
D	Блокировка	-	Блокировка доступа к данным на запись
темп.	Данные	(Время, (темп. кан. 1, темп. кан. 2))	Последние показания температурных датчиков
массив темп.	Данные	Список **temp.**	Все сохранённые показания температурных датчиков
w	Переменная	Рац. число	Время ожидания следующего задания в очереди
z	Переменная	Задание	Следующее задание: кортеж с событием, запросом к ТРИД, аргументами запроса (словарём) и контейнером для возврата результата

I	II	III	IV
r	Переменная	Пара: (флаг ошибки, результат запроса)	Результат запроса к ТРИД. Если запрос был успешен, то возвращается рац. число, пара рац. чисел либо **None**. В случае неудачи возвращается информация об ошибке.

Рисунок В.1 — Блок-схема потока взаимодействия с ТРИД, часть 1: получение основных настроек

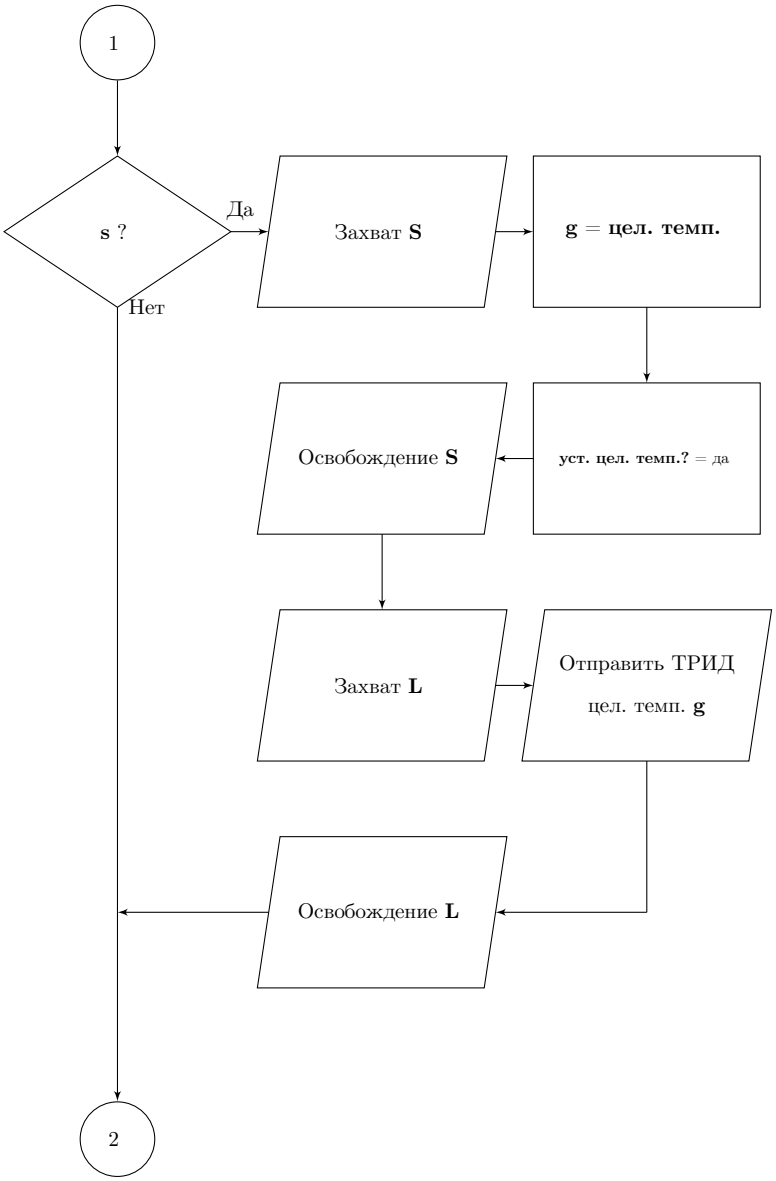

Рисунок В.2 — Блок-схема потока взаимодействия с ТРИД, часть 2: установка целевой температуры

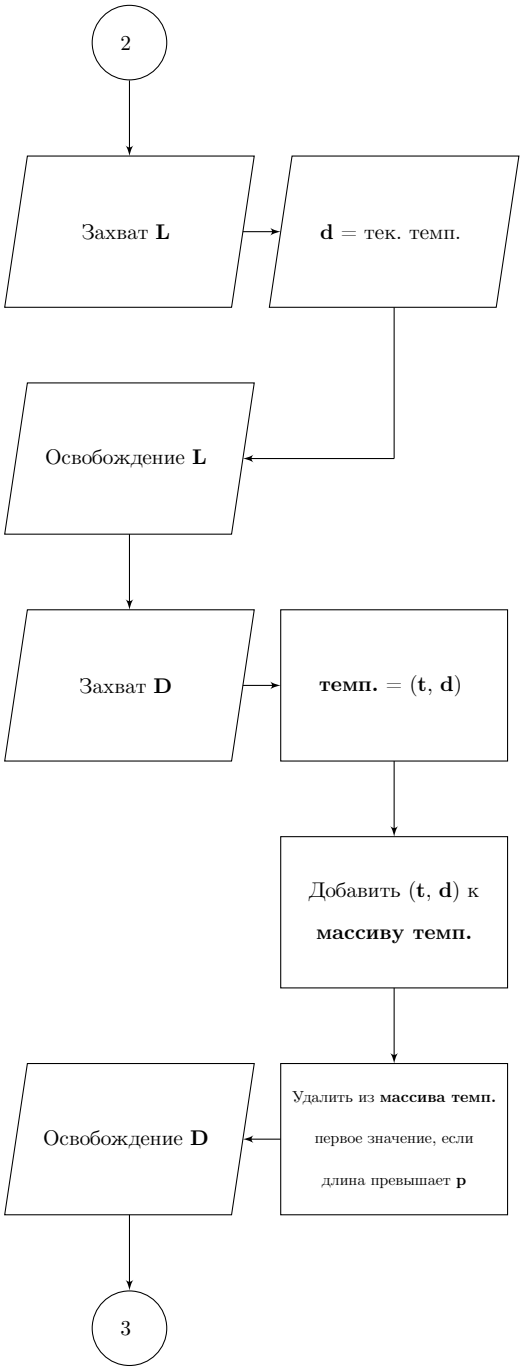

Рисунок В.3 — Блок-схема потока взаимодействия с ТРИД, часть 3: считывание и сохранение показаний датчиков температуры

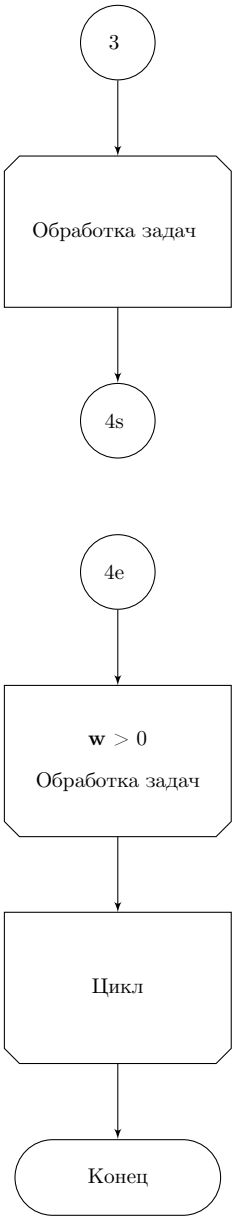

Рисунок В.4 — Блок-схема потока взаимодействия с ТРИД, часть 4: цикл обработки задач (часть 1)

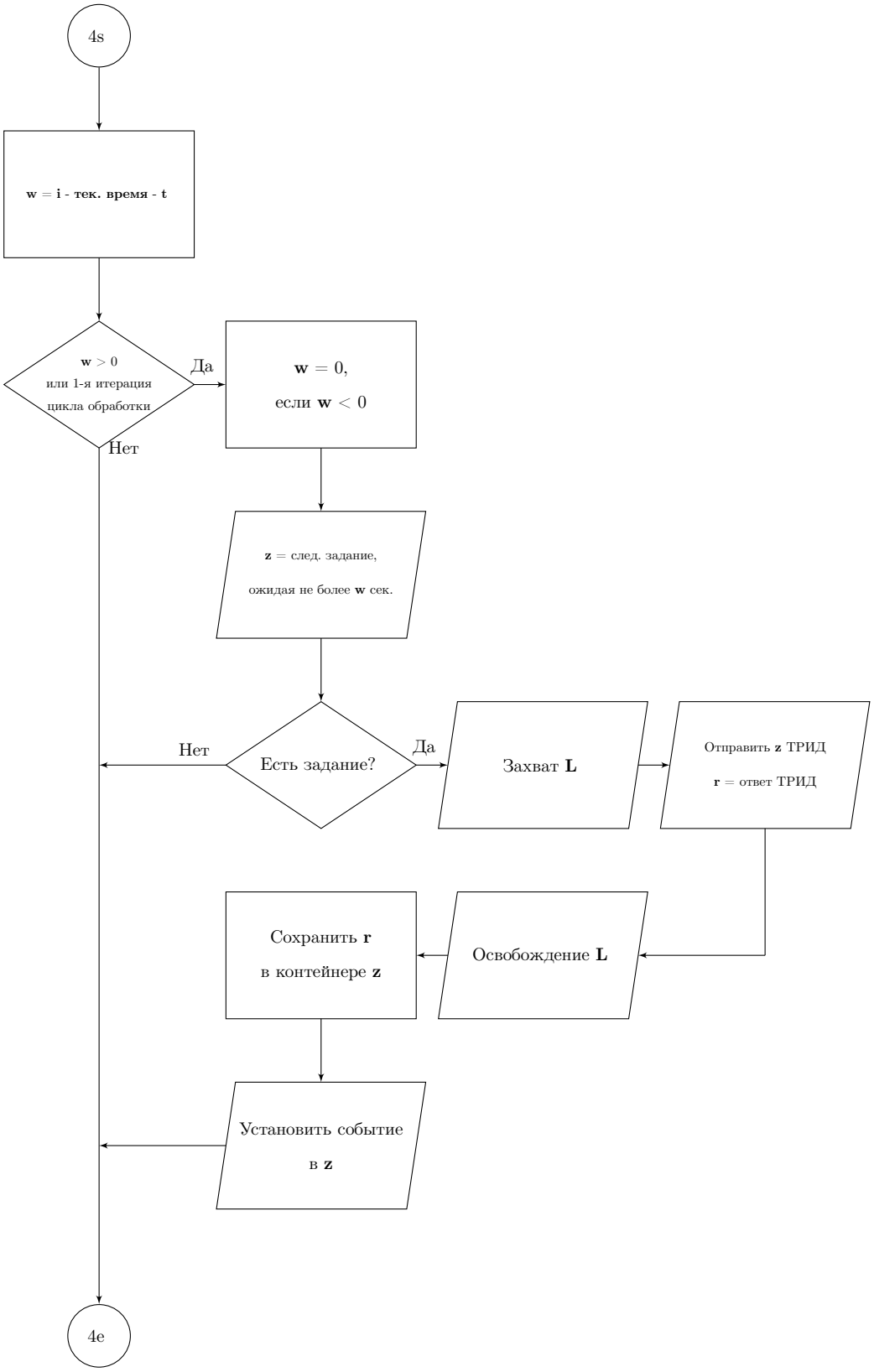

Рисунок В.5 — Блок-схема потока взаимодействия с ТРИД, часть 5: цикл обработки задач: основная схема

```
class PIDControllerThread(threading.Thread):
    '''PID regulator controller thread class

      This is the only way to access PID regulator: this makes calls to regulator
      synchronous.'''
    daemon = True

    def __init__(self, state):
        super(PIDControllerThread, self).__init__()
        self.__state = state

    def run(self):
        min_wait = 0.0
        while not self.__state.shutdown_event.is_set():
            start_time = monotonic()
            with self.__state.state_lock:
                interval = self.__state.interval
                data_points = self.__state.data_points
                did_set_target_temp = self.__state.did_set_target_temp
            self.__state.save_settings()
            try:
                if not did_set_target_temp:
                    with self.__state.state_lock:
                        target_temp = self.__state.get_target_temp()
                        self.__state.did_set_target_temp = True
                    with self.__state.trid_lock:
                        self.__state.trid.set_target_temp(values=target_temp)
                with self.__state.trid_lock:
                    temp = self.__state.trid.read_temperature()
            except Exception:
                logger.exception('Failed to get temperature')
            else:
                temperature = TimedTemperature(start_time, tuple(temp))
                with self.__state.data_lock.write_access:
                    self.__state.data.temperature = temperature
                    samples = self.__state.data.samples
                    if (len(samples) >= data_points):
                        del samples[0:len(samples) - data_points]
                    samples.append(temperature)
            first = True
            while True:
                wait_timeout = interval - (monotonic() - start_time)
                if wait_timeout <= min_wait:
                    if first:
```

```
                    wait_timeout = 0
                else:
                    break
            first = False
            try:
                finish_evt, meth, kwargs, ret_container = (
                    self.__state.job_queue.get(timeout=wait_timeout))
            except Queue.Empty:
                break
            else:
                try:
                    with self.__state.trid_lock:
                        ret = getattr(self.__state.trid, meth)(**kwargs)
                except Exception:
                    logger.exception('Caught exception when processing job')
                    ret_container.append((False, (sys.exc_info()[1:])))
                else:
                    ret_container.append((True, ret))
                finish_evt.set()
        self.__state.shutdown_event.wait(
            max(interval - (monotonic() - start_time), min_wait))
```

Листинг 14 — Реализация потока взаимодействия

Таблица Г.1 — API методы

Метод (без /api/)	Тип[1]	Описание	Параметры[2]	Возвращаемое значение[3]
I	II	III	IV	V
target/get	GET	Получить значение(я) целевой температуры.	channel	Словарь с ключами value1, value2, enabled1, enabled2
target	POST	Установить значение(я) целевой температуры и включить/выключить нагрев.	value1, value2, enabled1, enabled2	null
hyst/get	GET	Получить значение(я) ширины гистерезиса.	channel	Словарь с ключами value1, value2
hyst	POST	Установить значение(я) ширины гистерезиса.	value1, value2	null
alarm/a/get	GET	Получить значение аварийной температуры на канале A.	channel	Словарь с ключами value1, value2
alarm/a	POST	Установить значение аварийной температуры на канале A.	value1, value2	null
coef/kp/get	GET	Получить значение коэффициента K_p.	channel	Словарь с ключами value1, value2
coef/kp	POST	Установить значение коэффициента K_p.	value1, value2	null
coef/ki/get	GET	Получить значение коэффициента K_i.	channel	Словарь с ключами value1, value2

[1]GET, POST: тип HTTP запроса.

[2]См. табл. Г.2

[3]См. табл. Г.3 для возвращаемых «словарей», null означает, что при отсутствии ошибки возвращается строка null.

I	II	III	IV	V
coef/ki	POST	Установить значение коэффициента K_i.	value1, value2	null
coef/kd/get	GET	Получить значение коэффициента K_d.	channel	Словарь с ключами value1, value2
coef/kd	POST	Установить значение коэффициента K_d.	value1, value2	null
temp/get	GET	Получить показания датчиков температуры	channel	Кортеж с двумя значениями: **[timestamp, value1]**, **[timestamp, value2]** или **[timestamp, [value1, value2]]**.
temp/samples_since	GET	Получить список показаний датчиков температуры за данный период.	channel, timestamp	Список кортежей, идентичных кортежам, возвращаемым /api/temp/get.
temp/clear	POST	Сбросить сохранённый список показаний датчиков температуры.	нет	null
trid/connect/get	GET	Получить настройки соединения.	нет	Словарь с ключами mode, baudrate, slaveaddress, parity, stopbits, bytesize, port
trid/connect/list_ports	GET	Получить список известных портов UART, присутствующих в системе.	нет	Список строк.

I	II	III	IV	V
trid/connect	POST	Установить настройки соединения с ТРИД.	port, baudrate, slaveaddress, parity, stopbits, bytesize, mode	null
trid/monitor/get	GET	Получить настройки опроса ТРИД.	нет	Словарь с ключами interval, data_points
trid/monitor	POST	Установить настройки опроса ТРИД.	interval, data_points	null

Таблица Г.2 — Описание параметров, используемых в API

Название параметра	Значения параметра	Описание параметра
I	II	III
channel	1, 2	Выбирает канал, значение на котором интересует вызывающего. В зависимости от этого в ответе будут присутствовать ключи value1 или value2, enabled1 или enabled2. Если данный параметр отсутствует, то в ответе будут присутствовать оба значения.
value1	Рациональное число	Значение для канала 1.
value2	Рациональное число	Значение для канала 2.
enabled1	true или false	Включение/отключение нагрева для канала 1.
enabled2	true или false	Включение/отключение нагрева для канала 2.
timestamp	Временная метка (рациональное число)	При запросе части собранных значений: минимальное время, после которого находятся интересные значения. При отсутствии параметра возвращаются все значения.

I	II	III
. port	Строка, одна из строк из списка /api/trid/connect/list_ports	UART порт, используемый для подключения к ТРИД.
baudrate	Натуральное число, одно из 9 600, 19 200, 28 800, 57 600, 115 200	Скорость соединения с ТРИД.
slaveaddress	Натуральное число от 1 до 255	Адрес ТРИД.
parity	N, E или O	Чётность.
bytesize	7 или 8	Размер одного слова, пересылаемого по UART.
stopbits	1 или 2	Количество бит, означающих конец слова.
mode	asc или rtu	Режим протокола modbus: текстовый (asc) или бинарный (rtu)
interval	Неотрицательное рациональное число	Интервал опроса ТРИД в секундах. Может быть нулём, в этом случае ТРИД опрашивается с максимальной доступной скоростью с учётам необходимости дождаться ответа до посылки следующего запроса.
data_points	Натуральное число	Максимальное количество хранимых значений температуры.

Таблица Г.3 — Описание ключей словарей, используемых в возвращаемых значениях API

Название ключа	Значения	Описание
I	II	III
value1	Рациональное число	Значение для канала 1.
value2	Рациональное число	Значение для канала 2.
enabled1	true или false	Включение/отключение нагрева для канала 1.
enabled2	true или false	Включение/отключение нагрева для канала 2.

I	II	III
port	Строка, одна из строк из списка /api/trid/connect/list_ports	UART порт, используемый для подключения к ТРИД.
baudrate	Натуральное число, одно из 9 600, 19 200, 28 800, 57 600, 115 200	Скорость соединения с ТРИД.
slaveaddress	Натуральное число от 1 до 255	Адрес ТРИД.
parity	N, E или O	Чётность.
bytesize	7 или 8	Размер одного слова, пересылаемого по UART.
stopbits	1 или 2	Количество бит, означающих конец слова.
mode	asc или rtu	Режим протокола modbus: текстовый (asc) или бинарный (rtu)
interval	Неотрицательное рациональное число	Интервал опроса ТРИД в секундах. Может быть нулём, в этом случае ТРИД опрашивается с максимальной доступной скоростью с учётам необходимости дождаться ответа до посылки следующего запроса.
data_points	Натуральное число	Максимальное количество хранимых значений температуры.

Таблица Г.4 — Экспериментально определённые граничные значения value1 и value2 ключей и параметров

API метод	Единицы измерения	Шаг	Минимум	Максимум
I	II	III	IV	V
/api/temp	℃	0,1	−270	2 500
/api/target	℃	0,1	−200	2 500
/api/hyst	℃	0,1	0,1	50
/api/alarm	℃	0,1	−200	2 500
/api/coef/kp	℃	0,1	0	3 000
/api/coef/ki	секунда	1	0	9 999
/api/coef/kd	секунда	0,1	0	999,9

```
class HysteresisGetSetAPI(TRIDGetSetAPI):
    channel = '/api/hyst'
    _trid_suffix = 'hyst'
    _trid_kwargs = {}
class AlarmAGetSetAPI(TRIDGetSetAPI):
    channel = '/api/alarm/a'
    _trid_suffix = 'alarm'
    _trid_kwargs = {'alarm': 'A'}
class PIDCoefKpGetSetAPI(TRIDGetSetAPI):
    channel = '/api/coef/kp'
    _trid_suffix = 'pid_coef'
    _trid_kwargs = {'coef': 'Kp'}
class PIDCoefKiGetSetAPI(TRIDGetSetAPI):
    channel = '/api/coef/ki'
    _trid_suffix = 'pid_coef'
    _trid_kwargs = {'coef': 'Ki'}
class PIDCoefKdGetSetAPI(TRIDGetSetAPI):
    channel = '/api/coef/kd'
    _trid_suffix = 'pid_coef'
    _trid_kwargs = {'coef': 'Kd'}
```

Листинг 15 — Код всех используемых классов-потомков **TRIDGetSetAPI**

```
import json
from circuits.web import Controller
import circuits.web.exceptions as cwe

class TRIDConnectAPI(Controller):
    channel = '/api/trid/connect'

    def __init__(self, thread_state, *args, **kwargs):
        super(TRIDConnectAPI, self).__init__(*args, **kwargs)
        self.__thread_state = thread_state

    def get(self):
        '''Get currently used settings'''
        with self.__thread_state.trid_lock:
            trid = self.__thread_state.trid
            port_settings = trid.port_settings
            settings = {
                'mode': trid.mode,
                'baudrate': trid.baudrate,
```

```
20              'slaveaddress': trid.slaveaddress,
21              'parity': port_settings.parity,
22              'stopbits': port_settings.stopbits,
23              'bytesize': port_settings.bytesize,
24              'port': self.__thread_state.trid_port,
25          }
26      return json.dumps(settings)
27
28  def list_ports(self):
29      '''List available ports'''
30      return json.dumps(list(PORTS.keys()))
31
32  def POST(self,
33           port='pi-uart',
34           baudrate=trid.DEFAULT_BAUD_RATE,
35           slaveaddress=trid.DEFAULT_SLAVE_ADDRESS,
36           parity=trid.DEFAULT_PORT_SETTINGS.parity,
37           stopbits=trid.DEFAULT_PORT_SETTINGS.stopbits,
38           bytesize=trid.DEFAULT_PORT_SETTINGS.bytesize,
39           mode=trid.DEFAULT_MODE):
40      '''Connect to PID regulator using different settings
41
42          :note: Uses default settings if not specified, not last used
43              settings.'''
44      if port not in PORTS:
45          raise cwe.BadRequest(json.dumps({'error': 'Unknown port'}))
46      try:
47          baudrate = int(baudrate)
48          slaveaddress = int(slaveaddress)
49          stopbits = int(stopbits)
50          bytesize = int(bytesize)
51      except ValueError as e:
52          raise cwe.BadRequest(json.dumps({
53              'error': 'Argument not an integer'}))
54      if baudrate not in trid.SUPPORTED_BAUDRATES:
55          raise cwe.BadRequest(json.dumps({'error': 'Unsupported baud rate'}))
56      if not (0x1 <= slaveaddress <= 0xFF):
57          raise cwe.BadRequest(json.dumps({'error': 'Unsupported address'}))
58      if bytesize not in {7, 8}:
59          raise cwe.BadRequest(json.dumps({'error': 'Unsupported byte size'}))
60      if parity not in set('NEO'):
61          raise cwe.BadRequest(json.dumps({'error': 'Unsupported parity'}))
62      port_settings = trid.PortSettings(bytesize=bytesize, parity=parity,
63                                        stopbits=stopbits)
64      if port_settings not in trid.SUPPORTED_PORT_SETTINGS:
```

```
            raise cwe.BadRequest(json.dumps({
                'error': 'Unsupported port settings'}))
        if mode not in trid.MODE_TRANSLATIONS:
            raise cwe.BadRequest(json.dumps({'error': 'Unsupported mode'}))
        self.__thread_state.trid_connect(
            port,
            mode=mode, slaveaddress=slaveaddress, baudrate=baudrate,
            port_settings=port_settings)
        return 'null'
```

Листинг 18 — Код класса **TRIDConnectAPI**, обрабатывающего запросы к **/api/trid/connect**

```
import json
from circuits.web import Controller
import circuits.web.exceptions as cwe

class TRIDMonitorAPI(Controller):
    channel = '/api/trid/monitor'

    def __init__(self, thread_state, *args, **kwargs):
        super(TRIDMonitorAPI, self).__init__(*args, **kwargs)
        self.__thread_state = thread_state

    def POST(self, interval=None, data_points=None):
        '''Set monitor parameters: interval and amount of data points

            :param interval: Floating point value of new interval, in seconds.
            :param data_points: Integral value representing maximal amount of data
                points.'''
        if interval is not None:
            try:
                interval = float(interval)
            except ValueError:
                raise cwe.BadRequest(json.dumps({
                    'error': 'Expected floating-point value'}))
        if data_points is not None:
            try:
                data_points = int(data_points)
            except ValueError:
                raise cwe.BadRequest(json.dumps({
                    'error': 'Expected integral value'}))
        if (interval is not None and interval < 0) or (
                data_points is not None and data_points <= 0):
```

```
32              raise cwe.BadRequest(json.dumps({
33                  'error': 'Expected positive value'}))
34          with self.__thread_state.data_lock.write_access:
35              if interval is not None:
36                  self.__thread_state.interval = interval
37              if data_points is not None:
38                  self.__thread_state.data_points = data_points
39              self.__thread_state.need_save_settings = True
40          return 'null'
41
42      def get(self):
43          '''Get currently used interval value and number of data points'''
44          with self.__thread_state.data_lock.read_access:
45              ret = {'interval': self.__thread_state.interval,
46                     'data_points': self.__thread_state.data_points}
47          return json.dumps(ret)
```

Листинг 19 — Код класса **TRIDMonitorAPI**, обрабатывающего запросы к **/api/trid/monitor**

```
import bisect
import json
from circuits.web import Controller
import circuits.web.exceptions as cwe

PI_UART_PORT = '/dev/ttyAMA0'
'''Port used by UART on Raspberry Pi'''

PORTS = {'pi-uart': PI_UART_PORT}
'''Supported ports

Modify this dictionary to add support for other ports.'''

class TemperatureAPI(Controller):
    channel = '/api/temp'

    def __init__(self, thread_state, *args, **kwargs):
        super(TemperatureAPI, self).__init__(*args, **kwargs)
        self.__thread_state = thread_state
        self += TemperatureClearAPI(thread_state)
    def get(self, channel=None):
        if channel not in {None, '1', '2'}:
            raise cwe.BadRequest(json.dumps({'error': 'Unexpected channel'}))
        with self.__thread_state.data_lock.read_access:
            ret = self.__thread_state.data.temperature
        if channel is not None:
            ret = (ret.timestamp, ret.temperature[int(channel) - 1])

    def samples_since(self, channel=None, timestamp=None):
        '''Get samples since the given time

           Returns all samples if time is not given.'''
        if channel not in {None, '1', '2'}:
            raise cwe.BadRequest(json.dumps({'error': 'Unexpected channel'}))
        if timestamp is not None:
            try:
                timestamp = float(timestamp)
            except ValueError:
                raise cwe.BadRequest(json.dumps({'error': 'Expected float'}))
        with self.__thread_state.data_lock.read_access:
            ret = self.__thread_state.data.samples[:]
        if timestamp is not None:
            temp = TimedTemperature(timestamp, (float('inf'), float('inf')))
            idx = bisect.bisect(ret, temp)
            del ret[:idx]
        if channel is not None:
            tidx = int(channel) - 1
            ret = [(i.timestamp, i.temperature[tidx]) for i in ret]
        return json.dumps(ret)
```

```
from circuits.web import Controller
class TemperatureClearAPI(Controller):
    '''/api/temp/clear API entry point

      :param PIDControllerThreadState thread_state:
          PID controller thread state.'''
    channel = '/api/temp/clear'

    def __init__(self, thread_state, *args, **kwargs):
        super(TemperatureClearAPI, self).__init__(*args, **kwargs)
        self.__thread_state = thread_state

    def POST(self):
        with self.__thread_state.data_lock.write_access:
            self.__thread_state.data.samples[:] = []
        return 'null'
```

Листинг 17 — Код класса **TemperatureClearAPI**, обрабатывающего запросы к **/api/temp/clear**